Program Reference Manual

The TNT Refinement Package

Dale E. Tronrud

This manual is consistent with Release 5-F of TNT. Detailed information about the package and methods used can be found in the following references. These papers can also be viewed at

http://www.uoxray.uoregon.edu/dale/

- Tronrud, D. E., Ten Eyck, L. F., & Matthews, B. W. (1987). An Efficient General-Purpose Least-Squares Refinement Program for Macromolecular Structures. *Acta Crystallogr A,* **43**, 489–501.
- Tronrud, D. E. (1992). Conjugate-Direction Minimization – An Improved Method for the Refinement of Macromolecules. *Acta Crystallogr A,* **48** (November), 912–916.
- Tronrud, D. E. (1996). Knowledge-Based B-Factor Restraints for the Refinement of Proteins. *J App Cryst,* **29** (2), 100–104.
- Tronrud, D. E. (1997). The TNT Refinement Package. in Macromolecular Crystallography, Part B, Eds Charlie Carter, and Robert Sweet, Volume **277** in Methods in Enzymology, pp 306-319.
- Tronrud, D. E. (1999). The Efficient Calculation of the Normal matrix in lease-squares refinement of macromolecular Structures. *Acta Crystallogr A,* **55**, 700–703.

Up-to-date information about TNT can be found at

http://www.uoxray.uoregon.edu/tnt/

Contents

Acknowledgements

The general design of this package was initiated by Lynn Ten Eyck. He set up the user interface to the programs and established their internal data structures while here at Brian Matthews' lab. While Lynn completed much of the coding for the initial version of the program, he left the lab before the completion of the project. Mike Schmid picked up the project and got the programs running well enough that he could do some refinement but the programs did not behave properly and it was clear that much debugging and development was still required. In the process of this debugging and development a number of new features were incorporated into the package to enhance its flexibility, speed, generality, user-friendliness, and general usefulness as an all-purpose program for macromolecular structure refinement.

Since I picked up the project in 1981 I was aided in my work by many discussions with everyone in the Matthews lab. The most important and frequent of these discussions were with Meg Holmes, Larry Weaver, Jim Remington, and of course, Brian himself. Meg wrote the first draft of this manual, and most everyone in the lab has proofread it at one time or another.

The suggestion that the gradient/curvature method be incorporated into TNT was first made by Ron Stenkamp.

Conventions

There are a number of conventions used in this document. One set of conventions use the typeface to indicate what kind of object is being described. These conventions are

Bold	This is the name of a script, e.g. **tnt**.
`Teletype`	This is a file name, e.g. `tnt/data/formfactor.dat`. This typeface is also used to display exact dialogs between the user and the computer. In such an example the text typed by the user is in bold.
First letter capitalized	This is the name of a program, e.g. Rfactor.

Another series of conventions are involved in the specification of syntax. When describing a data statement or command there has to be a method to specify that something is optional or something else can be repeated. These conventions are

< . . . >	Substitute a value. The text within the broken brackets describes the value. In the input line the broken brackets are not entered.
N(. . .)	Repeat the contents of the parenthesizes zero or more times.
[. . .]	The contents of the brackets are optional.
{ . . . | . . . }	Choose one of the options separated by "|". There may be more than two options to choose from.

Chapter 1

TNT Refinement Package

Congratulations! You have decided to leave the world of canned shell scripts and enter the real world of TNT operation.

TNT is actually a collection of utility programs which can be ordered and combined in many different ways. The scripts you have been using are one way of putting the pieces together but there are many uses and functions in TNT which are not accessible from the limited world of rigid scripts.

The design of TNT differs from that of other refinement packages. It is broken into a number of isolated programs, each of which performs a specific and limited set of functions. All of the programs are similar in that they enjoy the same style of input and share a great deal of internal code. They only differ in the particular command statements each will accept.

Originally this design was chosen to save memory and improve the ability to handle special cases. It did not make sense to allocate memory for holding structure factors when evaluating stereochemical functions or to allocate memory to store the standard geometry library when calculating structure factors. Since Fortran 77 does not allow dynamic allocation of memory the only way to control this is to write separate programs.

In addition this architecture was chosen because there was no means to handle the numerous space groups efficiently without using a different program for each one. In the far past one calculated structure factors by first running a space group general program to calculate electron density maps from coordinates, then running a space group specific program which read the map file and calculated structure factors, and finally running the original form of the program Rfactor which read these calculated structure factors. In the current form of TNT all these activities are incorporated in the libraries which underlie all the crystallographic programs.

One benefit of TNT's design which is still quite relevant is that one can access the data which flows from one program to another. If you are developing your own ideas for refinement you can work them into TNT without learning its internal structure and you can optimize your calculations by structuring your data in whatever way you want.

Clearly the direction of TNT is to be merged into larger programs until the entire package is unified. The challenge of design is to find a way to bring these programs together while maintaining your ability to include your own ideas. That is my job.

The Programs of TNT

The principle programs of TNT are:

Geometry

Geometry is the TNT program which performs all tasks related to standard stereochemistry. Its basic function in refinement is to read a coordinate file, along with its corresponding sequence file and standard geometry library, to calculate the function value, gradient, and curvature of the stereochemical residual function. Its other great task is to produce the list of worst violations of each kind of stereochemistry. Both of these tasks are accessed via the shell scripts **tnt** and **geometry**.

Geometry also has a very useful tool for the evaluation of the quality of a standard library. One can read a coordinate file and a library and, instead of looking for outliers, look at the standard values the library would have to have to match the coordinates. If one has a number of high resolution models they can all be read into Geometry and its list can be checked for systematic differences between the coordinates and the library. This list is generated with the `$tntbin/report_geo` script.

Rfactor

Rfactor is used in any case where two sets of structure factors are compared and scaling is required. This is in contrast to Fourier which is used whenever a single set of structure factors are used or two sets are used without scaling. If you want to calculate difference maps of various sorts this is the place to go.

In addition, Rfactor performs very complicated calculations in its computation of the function value, first, and second derivatives of the crystallographic restraining function. Rfactor has several commands which allow you to directly access portions of these calculations and subvert them to your will. The AGARWAL command allows you to supply special Fourier coefficients to calculate a gradient. The NORMAL_MATRIX command allows you to, likewise, perform special purpose calculations of the diagonal of the Normal matrix. No shell scripts are supplied to access these commands.

NCS

The program NCS handles the constraint or restraint of noncrystallographic symmetry. It is well encapsulated by the shell scripts supplied with the TNT distribution (`$tntbin/tnt_cycle_ncs`, **ncs**, **gather**, and **scatter**). To constrain a model to noncrystallographic symmetry you need to modify a refinement script to meet your needs. A prototypical script is supplied in `tnt/doc/tnt_cycle_con_ncs` and the required changes are described in the "Defining Noncrystallographic Symmetry" chapter of the TNT Users' Guide.

Shift

The program Shift contains all the code which implements the function minimizer of TNT. Shift does not know anything about diffraction, stereochemistry, or noncrystallographic symmetry. It is quite flexible about what function it is minimizing. The form and definition of the parameters to be varied are built into Shift and these cannot be changed. TNT will allow you to change the data your model is restrained to but will not allow you to change the nature of the parameters themselves. (Someday this may change.)

Convert

Convert reads and writes many coordinate file formats. Only conversions for the Protein Data Bank (PDB) format and Alwyn Jones' DSN2 format can be performed using shell scripts (**to_pdb**, **from_pdb**, **to_dsn2**, and **from_dsn2**). You can write your own scripts to access the other formats.

Fourier

Fourier is basically a file format converter for crystallographic data files. This view is based on the point of view that Fourier coefficients and density maps are merely different formats of the same information. A Fourier transform is required to convert the data from one representation to the other but the information is conserved in the process. Since all TNT programs which read crystallographic data can perform Fourier transforms themselves and will accept either a set of Fourier coefficients or a map interchangeably this use of Fourier is not usually needed.

The other task of Fourier is to perform various operations on crystallographic data sets. It is the program used by the **correct** and **report_hkl** scripts. A set of functions of Fourier not accessible from the shell scripts is its ability to add, subtract, multiply, or divide any pair of crystallographic data sets. For example, you can ask for a map which is determined from the sum of two sets of Fourier coefficients. In fact you can add a map and a set of coefficients and the program will Fourier transform one of them and then add them.

How do I . . . ?

- calculate a difference map?

  A difference map is usually some function of both Fo and Fc. Since scaling of the two structure factor sets is required you need to use the program Rfactor. If all you want is a 2Fo-Fc or Fo-Fc map you can use the **make_maps** shell command. If you want some other map you should take the script `$tntbin/make_maps` and edit a copy to change the type of the Fourier coefficients to those you want.

- calculate a Fourier transform?

  Fourier transforms are calculated with the program Fourier. If you want to calculate a map from coefficients simply PUNCH a map using the coefficient file as the "source". If you want the other direction PUNCH the coefficient file with a map file as its source. Usually Fourier transforms do not need to be explicitly calculated in TNT. Any operation that requires crystallographic data will accept either a map or a set of coefficients as input. As coefficient files are smaller you should try to store data in that form whenever possible.

- change a map file format?

  The only map file formats supported in TNT are the traditional Ten Eyck map format and Alwyn Jones' DSN6 map format. TNT will not read a DSN6 format file. This means that the only conversion possible is to convert a Ten Eyck map file to DSN6. This is done with the program Fourier. Simply PUNCH the DSN6 format map using the Ten Eyck map as the "source". Such conversions are usually unnecessary because any TNT program that can write a Ten Eyck map file can write a DSN6 file directly.

- change a coordinate file format?

  To read or write PDB or DSN2 format coordinate files use the **from_pdb**, **from_dsn2**, **to_pdb**, or **to_dsn2** shell commands. You can also convert a PDB file directly to DSN2 using the `$tntbin/pdb2dsn2` script. For other file formats you need to write a script for the program Convert. You can use one of those scripts supplied in TNT as a template. Remember, one usually cannot convert file formats without some manual editing.

Chapter 2

Theory of TNT Refinement

The TNT package uses gradient descent methods to optimize the parameters of the molecular model. For methods of this type the structure of a cycle of refinement is the same regardless of the particular method used (Agarwal, R.C., Acta Cryst. (1978). **A34**, 791-809). Think of each parameter of the model as a component of a very long vector. First a new vector is chosen to add to the original vector, this is the shift vector. Only a fraction of this vector, however, is to be added. The second part of each cycle is to determine how much.

If the model vector is called $\mathbf{x}$ and the shift vector $\mathbf{s}$, the new model vector $\mathbf{x}'$ is

$$\mathbf{x}' = \mathbf{x} + \alpha\mathbf{s}$$

where α is the fraction of the shift to be applied. The two steps of a cycle of refinement are first to calculate $\mathbf{s}$ and then to determine α. The different minimization methods available in TNT alter how $\mathbf{s}$ is calculated. The calculation of α is always done the same way.

What Direction to Shift?

How does one calculate $\mathbf{s}$? TNT offers several options. The simplest method is derived by recognizing that the least we need is for $\mathbf{s}$ to point downhill. Since the gradient of a function points in the steepest uphill direction, by definition, the negative of the gradient will point in the steepest downhill direction. Equating **s** with the negative of the gradient of the function $(-\nabla f(\mathbf{x}))$ gives us the steepest descent method.

The steepest descent method is very good as long as the curvature for each parameter is the equal, but this is far from true in crystallographic refinement. Because of the difference in curvatures the minimum curves away from the direction of the gradient, and many, many successive cycles of refinement are required to reach the minimum. Some account must be made for the curvature.

The conjugate gradient method (Fletcher, Reeves, Computer Journal (1964). **7**, 149) uses the difference between the gradient in the last cycle and the current gradient to infer the curvature and derive a superior direction to shift. In effect, it learns the curvature of the function as more and more cycles are run and compensates for it. The conjugate gradient method requires the current gradient and the direction of shift from the last cycle. This means that it cannot be used in the first cycle of a series because in the first cycle there is no old direction of shift. Also you cannot change anything about the model between the two cycles. The weights, modules used, and scale factors must remain the same. If you need to change the weights, or something else, you must restart with a steepest descent cycle.

To summarize, with conjugate gradient minimization

$$\mathbf{s}_k = -\nabla f(\mathbf{x}_k) + \beta_k \mathbf{s}_{k-1} \tag{2.1}$$

$$\beta_k = \frac{\nabla f(\mathbf{x}_k)^t \nabla f(\mathbf{x}_k)}{\nabla f(\mathbf{x}_{k-1})^t \nabla f(\mathbf{x}_{k-1})} \tag{2.2}$$

Because the conjugate gradient method deduces the curvature from previous cycles it sometimes takes quite a few before it can overcome the problems introduced by large differences in curvature. For positional parameters, x, y, and z, the curvatures are close to the same magnitude for all atoms except when a particular atom has many more electrons than the rest. A heavy atom will tend to be overshifted each cycle which results in an oscillation. In steepest descent the oscillation problem is very bad and persists after many cycles. With conjugate gradient the oscillations will damp out with time but you must monitor the individual shifts of the heavy atoms to determine when convergence has occurred.

The B factors of heavy atoms also oscillate during refinement, however a more serious problem occurs with atoms with large B factors. The curvature for the B factor of these atoms is smaller than the average which causes the parameters to be undershifted. The B factors actually lock up with values smaller than optimal if they started too small and with values too large if they started too large. Since B factors usually start small the large B factors of a model refined with steepest descent or conjugate gradient are systematically underestimated.

The solution to these problems is to include explicitly the curvature of each parameter. The method which uses curvature that is equivalent to steepest descent is the gradient/curvature method. In this method the direction of shift, **s**, is defined to be the negative of the ratio of the gradient and the diagonal of the curvature matrix. In other words (so to speak)

$$\mathbf{s} = -\frac{\nabla f(\mathbf{x})}{\nabla_d^2 f(\mathbf{x})} \tag{2.3}$$

Given the same gradient, parameters with large curvatures are shifted smaller amounts while those with smaller curvatures are shifted more. The calculation of the shift vector with this method requires the gradient of the function and the diagonal of the curvature.

Besides improving the rate of convergence the explicit incorporation of the curvature allows the positional and thermal parameters to be refined at the same time. With steepest descent and conjugate gradient the large difference in curvature between the different types of parameters results in the requirement that they each be refined separately. First the XYZ parameters must be refined with the B factors held constant and then the B's are refined with the positions held. The process must be alternated several times to allow each class of parameter to adjust to the changes in the other. With gradient/curvature all parameters may be varied in each cycle.

While the incorporation of the diagonal part of the curvature improves convergence a great deal the off-diagonal part is still not used.

The explicit use of these elements of the normal matrix would consume a great deal of computer time and require extensive modifications to the structure of TNT. If a method equivalent to conjugate gradient existed which used the gradient/curvature method as its base instead of steepest descent the off-diagonal elements would be “learned” from the history of the refinement. Such a method has been devised (Axelsson, O., Barker, V.A., *Finite Element Solution of Boundary Value Problems* (1984) Chapter 1, Academic Press, Inc), (Tronrud, D. E., Acta Cryst. (1992) **A48**, 912-916) and called preconditioned conjugate gradient. To calculate the shift vector requires the gradient and curvature of the function as well as the shift vector from the last cycle. Like conjugate gradient nothing may be changed between successive cycles of preconditioned conjugate gradient refinement. If the weights are to be changed refinement must begin again with gradient/curvature.

The equations for the shift in TNT’s version of preconditioned conjugate gradient are

$$\mathbf{s}_k = -\frac{\nabla f(\mathbf{x}_k)}{\nabla_d^2 f(\mathbf{x}_k)} + \beta_k \mathbf{s}_{k-1} \tag{2.4}$$

$$\beta_k = \frac{\nabla f(\mathbf{x}_k)^t \nabla_d^2 f(\mathbf{x}_k) \nabla f(\mathbf{x}_k)}{\nabla f(\mathbf{x}_{k-1})^t \nabla_d^2 f(\mathbf{x}_k) \nabla f(\mathbf{x}_{k-1})} \tag{2.5}$$

How Far to Shift?

Once the shift vector has been calculated the fraction of this vector, or α must be determined. We want to find a value for α which will cause $f(\mathbf{x} + \alpha\mathbf{s})$ to be as small as possible. Since we know $\mathbf{x}$ and now know $\mathbf{s}$ the problem is a one dimensional minimization problem. The strategy used in TNT to find α is

1. Guess a value for α.
2. Calculate $f(\mathbf{x} + \alpha\mathbf{s})$.
3. Fit a parabola to the known data.

4. Set α to the minimum of the parabola.

5. Goto 2

The process repeats until the change in α from one cycle to the next is insignificant.

The particular definition of insignificant used is "When the predicted improvement in function value is less than five percent". The function value at the point $\mathbf{x} + \alpha\mathbf{s}$ was smaller than it was at the start. The function value should be even lower with the new value of α. From the parabola we can predict how much lower the function value will be at the new point. This difference is the predicted improvement in function value. If the predicted improvement is small there is little reason to deliberate the merits of the different values for α. It is time to start an entirely new cycle.

The one dimensional minimization loops require the calculation of the value of the function but no gradients or curvatures. They take a small proportion of the computer time of the entire cycle of refinement. Therefore they are called "short loops". In a fit of bad analogy, the name "long loop" has been given to the portion of a cycle which calculates the shift vector. A cycle of refinement consists of a single long loop and two short loops. If the approximations of the calculations are violated in some fashion more short loops will be required. If additional short loops are executed there may be nothing wrong with the cycle but it should be examined for error.

Least Squares Refinement

The function minimized in least-squares optimization is

$$f(\mathbf{x}) = \sum_i \frac{1}{\sigma_o^2(i)} (Q_o(i) - Q_c(i, \mathbf{x}))^2$$

Where $Q_o(i)$ is the i(th) observation, known with the standard deviation $\sigma_o(i)$, and $Q_c(i, \mathbf{x})$ is the corresponding prediction of the model given a set of parameters, $\mathbf{x}$. If the model is being restrained by several types of observations it is more convenient to use a separate term for each type. Then the function becomes

$$f(x) = f_1(\mathbf{x}) + f_2(\mathbf{x}) + f_3(\mathbf{x}) + \cdots .$$

Where f_1 is the sum over the first class of observations and so forth.

In the long loop we want to calculate $f(\mathbf{x})$ as well as its gradient and usually its curvature. The first and second derivative operators are both linear, which means that the first derivative of a sum is the sum of the first derivative of each term. We can calculate the gradient and curvature for each type of observation separately and add them together later.

Using this property TNT has been broken into separate programs. There is a different program for each term of the function which calculates the function value, gradient, and curvature for their term. The central control program, Shift, combines the information for each term and performs the tasks which coordinate the cycle of refinement. The separation of the calculations into programs by observation type allows new data types to be used as restraints, or existing ones to be ignored, without modifying the programs which handle the minimization itself. To add a new type of observation you simply write a program which can calculate the value of that term and its gradient. If possible it should also calculate the curvature so that the more powerful methods of function minimization can be used.

The Refinement Script

TNT is supplied with scripts for performing the most common types of refinement. These scripts are listed in Table 2.1. The simplest means to develop a new script is to modify one of these files. To understand the data flow within a TNT refinement job you must understand the file naming convention and the way the different terms in the equation are implemented in the programs of TNT.

The input to a cycle of refinement is the starting model, named by convention `init.cor`, and the optional old shift vector named `olddir-.dat`. Because the results of each cycle is usually fed into the next the

tnt_cycle_regularize	stereochemistry alone
tnt_cycle	diffraction data and stereochemistry (default)
tnt_cycle_ncs	ncs symmetry as well
tnt_cycle_realspace	real space refinement w/ geometry
tnt_cycle_phase	amplitude, phase, and geometry

Table 2.1: Scripts Supplied with TNT.

output of a cycle of refinement is the same type of data — a coordinate file and a shift vector. However the new files have different names: `shifted.cor` and `newdir.dat`. The code provided by these scripts renames `shifted.cor` to `init.cor` and `newdir.dat` to `olddir.dat` at the end of each cycle.

Refinement Script Details

The first half of the script is the long loop. The program for each term is run to calculate the function value and gradient using the parameters in `init.cor`. If you want to run steepest descent or conjugate gradient this is all the information you need, and you would use the GRADIENT command. If you wish to use gradient/curvature or preconditioned conjugate gradient, which are the preferred methods, you would use the CURVATURE command instead.

Therefore each program is provided with the observations to which the model is to be restrained, the coordinate file (`init.cor`), and finally given the CURVATURE command. (I suggest you read `$tntbin/tnt_cycle` at this point.)

The control program, Shift, is given the files produced by each module as well as `init.cor` and `olddir.dat`. The flow of data in a long loop is shown diagrammatically in Figure 2.1.

If you do not have an old shift vector or do not want to use one of the conjugate gradient methods you should still INCLUDE `olddir.dat` and simply supply an empty one. Having Shift read an `olddir.dat` regardless of whether one is there allows you to switch between conjugate and non-conjugate methods without editing the script. Switching between a gradient and a curvature method simply requires that the GRADIENT command be replaced with a CURVATURE command.

In the long loop Shift should be instructed how to vary the parameters. The definition of which parameters are to be held constant, such as occupancies, and which parameters are to be treated as collected groups, as in rigid body refinement or when portions of the model are to have single overall B factors, must be given to Shift. Since these definitions are usually given in the control file Shift should read this file as well.

In the short loop each module is ran once again (Figure 2.2). This time they are given the coordinate file `shifted.cor` and execute the FUNCTION command. Their output files are all read by Shift. The flow of control in the short loop can be seen in the standard script `$tntbin/tnt_cycle`.

Adding your Own Module

1. Choose a script as the starting point.
2. Run the new program in the long loop.
   (a) Have it read `init.cor`
   (b) Define the command to be either GRADIENT or CURVATURE with the $COMMAND environment variable.
3. Have Shift read the output of the new module
   (a) Remember to delete the file after Shift's run
4. Add the new program to the short loop.
   (a) Use `shifted.cor` this time.

```
olddir.dat   init.cor
    |         | | | |_______________________________________
    |         | | |_____________________                   |
(optional)    | |______              |                     |
    |         |       |              |                     |
    |         |  ------------   ------------------  ----------------
    |         |  |   X-Ray  |   | Stereochemical |  | User Defined |
    |         |  |  Module  |   |     Module     |  |   Modules    |
    |______   |  ------------   ------------------  ----------------
          |   |       |              |                     |
          |   |  rfactor.dat     geometry.dat           ???????
          |   |       |              |                     |
          |   |   ____|              |                     |
          |   |   |  ________________|                     |
          |   |   |  |  ___________________________________
          |   |   |  |  |
          |   |   |  |  |   <------ Gradient, function value
        -------------------         (, and optionally curvature)
        | Control Program |         from each module.
        -------------------
           |     |     |
stpfil.dat |     |     | newdir.dat
           |     |     |
                 |
                 | shifted.cor

            To Short Loop
```

Figure 2.1: Long Loop Data Flow

This figure shows the flow of data through a long loop and the standard file names. Each module contributes one set of derivatives and a function value for its term in the function being minimized. It may also supply the curvature.

The control program (Shift) produces three files. `newdir.dat` *is the direction file. This file contains the shift vector calculated by combining the information from each module.* `stpfil.dat` *is used to communicate with the next run of Shift in the short loop. It passes information about the history of the search for the minimum along the shift vector.* `shifted.cor` *is a coordinate file where the first guess as to what the best shift should be has been applied.*

```
init.cor  newdir.dat  stpfil.dat      shifted.cor
   |         |            |             |  |  |
   |_______  |  __________|             |  |  |
          |  |  |       ________________|  |  |_____________
          |  |  |       |               ___|              |
          |  |  |       |               |                 |
          |  |  |   ----------  ------------------  ----------------
          |  |  |   |  X-Ray |  | Stereochemical |  | User Defined |
          |  |  |   | Module |  |     Module     |  |   Modules    |
          |  |  |   ----------  ------------------  ----------------
          |  |  |       |              |                  |
          |  |  |   rfactor.dat   geometry.dat        ?????????
          |  |  |       |              |                  |
          |  |  |   _____              |                  |
          |  |  |   |  _________________                  |
          |  |  |   |  |  __________________________________|
          |  |  |   |  |  |
          |  |  |   |  |  |   <----- Function values
         -------------------         from each module
         | Control Program |
         -------------------
             |       |
             |       |
  stpfil.dat |       | shifted.cor
             |       |

         To Next Short Loop
```

Figure 2.2: Short Loop Data Flow

In a short loop the "goodness" of the present model is determined for each term in the function by the appropriate module. Each module produces a file containing the function value for its term This information is combined with information from the last short loop (`stpfil.dat`*) and the starting point (*`newdir.dat`*) and a value for α is calculated by the control program. The product, $\alpha\mathbf{s}$, is added to the starting coordinate set (*`init.cor`*) and is written out in a new* `shifted.cor`*. Also, information about this short loop (current total function value and α) is written out in a new* `stpfil.dat`*.*

   (b) Write only the function value.

5. Have Shift read the output file

   (a) Delete the file.

Chapter 3

Shared I/O Properties

All of the programs in TNT read their input in the same fashion. They were designed so that the order in which the data are read is unimportant. In addition the data statements, themselves, are free format. You never need worry about the number of spaces required between two words.

Properties of a Statement

Each statement is either empty or contains several tokens. A statement can consist of several lines of text. If you wish a statement to be continued on the next line simply end the line with a "\" character (or a dash ("—") with a blank space preceding it). The statement can only be broken between tokens, never in the middle of a word or number. The number of characters on a statement cannot total more than 1024. This limit is usually not a problem, but like any limit in TNT can be increased if need be.

If the "#" character is encountered (or "!" for VMS people) the rest of the line is presumed to be a comment. If the last letter prior to the beginning of the comment was a dash then the statement will be continued on the next line. One may have a statement spread over many lines, with a comment at the end of each.

A token is a collection of any characters other than spaces, tabs, commas, or control characters. Tokens can be empty, such as a comma with a blank before it or at the end of statement. An empty token usually denotes that a default value is to be used for that piece of information.

Tokens can contain either character or numeric data. Numbers may be entered in any format (including “E” format). Decimal points are not required when the value of the number is integral. When the token contains character data all letters will be capitalized before the data is interpreted (except for file names). This means that “ca” will be treated the same as “CA”.

While all the characters of a token containing a number are important, only the beginning of a text token is saved. For some tokens only the first four letters are saved, others the first eight. With file names the first 128 letters are saved. Whether a particular token is significant to four or eight letters depends on the data type of the information which is being specified.

All statements begin with a keyword which specifies the type of the statement. The first 8 letters of a keyword are significant. There are two classes of keywords: data keywords and command keywords. Data keywords begin statements that contain information while command keywords specify actions that the program should take before any more statements are read. Many data keywords are common to all programs.

File Properties

Initially input is read from the terminal (Fortran logical unit #5). In general, data will be entered by either typing it into the script for the batch job or placing it in a text file and using the INCLUDE command to bring it to the attention of the program.

You are allowed to arrange your data in any fashion you wish. You can enter it all directly in the script or can separate different types of data in different files. Some kinds of data are consistently stored in separate files — All the coordinates are stored together in a file and called "the coordinate file". However even this is not required. Sometimes it is useful to separate the coordinates into several files and bring them together at the last minute. Whatever the structure of the data TNT will interpret it the same way.

Standard TNT Data Statements

Many of the data keywords are the same for all programs in this package. In all cases the meaning and token layout will always be the same for all programs which read the statement.

To simplify some of the descriptions below two abbreviations are used. The definition of the first is:

$$< \text{Residue designator} >:== \begin{cases} < \text{Chain name} > \mid < \text{Residue name} > \\ < \text{Residue name} > \end{cases}$$

This construction is used to specify a particular residue of a particular chain. It is used both to refer to residues within a chain and to sequence entries within a chain type. The second form is used when one wishes to refer to a residue within the default chain (which has no name). For example, "A|123" designates residue 123 of chain A and "123" refers to residue 123 of the default (nameless) chain.

The second abbreviation is

$$< \text{Atom designator} >:==< \text{Atom} > < \text{Residue} > < \text{Chain} >$$

This abbreviation is used when a particular atom is to be specified. If the atom belongs in the "chain with no name" the <Chain> field is left blank.

```
ASSUME RESIDUE_TYPE <Name> WHEN_CONTAINS N(<Name>)
```

This statement allows you to define a default residue type for any residue which contains all of the atoms listed, and no more. The puddle of water that is contained in most coordinate files presents a continuing maintenance problem in TNT's sequence file. Without the ASSUME statement you would have to edit the sequence file every time you added or removed solvent.

The following example will, I hope, clarify this statement's use. If your model has a great deal of DMSO, instead of creating individual RESIDUE statements for each one you could add to your control file the statement

ASSUME RESIDUE_TYPE DMSO WHEN_CONTAINS S C1 C2 O

Built into TNT is the assumption that a residue which contains only an atom named "OH" is of type "HOH". This is the PDB convention for water molecules.

```
ATOM  <type> <X> <Y> <Z> <B> <Occ>  <Atom designator>
ATOMC <type> <X> <Y> <Z> <B> <Occ>  <Atom designator>
ATOMG <type> <X> <Y> <Z> <B> <Occ>  <Atom designator>
```

These statements specify the parameters for the particular atom named on the statement. The particular keyword used defines the coordinate system in which the position of the atom is specified. ATOM statements use fractional coordinates, ATOMC statements use an orthogonal system (x is along a^* — this is not the same convention used in PDB format files.) and ATOMG statements define the position in terms of map grid units. Internally, positions are stored in the orthogonal system. To allow for the coordinate conversion upon input, ATOM statements must be preceded by a CELL statement and ATOMG statements must be preceded by both a CELL statement and a GRID statement. For more information see Appendix A.

```
CELL <a> <b> <c> <Alpha> <Beta> <Gamma>
```

This statement gives the cell dimensions in Ångstroms and degrees. It is required in almost every program and giving one when not required does no harm so one should always give a CELL statement. Since ATOM and ATOMG statements require that a CELL statement proceeds them, it is best to always have the first statement in the data stream be a CELL statement.

```
CHAIN <Chain name> <Chain type>  N(<Chain linkage>)

<Chain linkage> :==
       <Residue name> <Residue designator> <Linkage type>
```

The CHAIN statement is used to specify the type of a particular chain and the locations and types of any links between this chain and other chains. The link is assumed to exist between the chain named on this statement and the nearest symmetry image of the target chain.

The link defined on this statement will most likely be disulfide bonds between separate polypeptides of an oligomer or connections to symmetry-related molecules.

For example, the statement

CHAIN A ALPHA 44 B|123 SS

says that chain A is of type ALPHA and that there is a SS (disulfide) link between A|44 and B|123. The actual image of chain B linked to may be in the same asymmetric volume as A or it may be a symmetry image of B. The program will decide which symmetry operator to use.

For a more detailed description of this statement see the "TNT Sequence File" chapter in the TNT Users' Guide.

```
FORMFACTOR  <Atom type>  <Form factor parameters>
```

This statement allows the user specify a form factor approximation for a particular atom type. The form factors for all the atom types present in your structure must be defined each time you run a program which calculates electron density (or structure factors) from the model. A list of the definitions for many atom types is provided in the file `$tntdata/formfactor.dat`.

The atom type name on the statement can be up to 4 characters long and must match the atom type name on the ATOMx statements. The equation used to fit the form factors of the atoms is the sum of three gaussians,

$$f(s) = A\exp(-Bs^2) + C\exp(-Ds^2) + E\exp(-Fs^2),$$

where s is $1/2\sin\theta/\lambda$. On the FORMFACTOR statement the numbers are given in the order of A, B, C, D, E, F. The first pair should be the pair with the largest exponential coefficient (e.g. $B > D > F$). If F is equal to zero then one has the form of the equation used by Lee and Pakes (Acta Cryst. (1969). **A25**, 712) and the parameters from their paper can be used. (The values used in TNT are not from this paper, however there is no problem using the paper's values together with the TNT numbers.) The temperature factor parameters, B, D, and F, cannot be negative.

Here are some examples:

```
FORMFACTOR  C  1.220272  46.308872  3.084011  15.230391  1.691442
FORMFACTOR  N  2.504548  27.074999  3.262855  6.722432   1.219796
FORMFACTOR  O  1.514483  27.753214  4.691878  8.545067   1.790204
```

These statements define the form factors for atom types "C", "N", and "O". The parameter F is not given so it is assumed to be zero.

```
GEOMETRY <Residue type> <Restraint type> -
                      <Ideal value> <Sigma> N(<Atom name>)
GEOMETRY <Linkage type> <Restraint type> -
                      <Ideal value> <Sigma> N(<Atom name>)
```

GEOMETRY statements are used to define geometry restraints for particular residue or linkage types. (See the "Standard Geometry Definition" chapter of the TNT Users' Guide.)

```
GRID <Grid X> <Grid Y> <Grid Z>
```

This statement gives the number of grid divisions per unit cell along the crystallographic axes x, y, and z. This line must be included if the coordinates are read in ATOMG format, or if an ATOMG coordinate file is to be created through the use of the PUNCH command. The GRID statement must precede any use of ATOMG statements.

```
LATTICE <Text>
```

This statement gives the name of the lattice centering operators. The codes to be used are listed in Table 3.1. The default, if no statement is given, is Primitive.

The proper LATTICE and OPERATOR statements for all useful space groups are given in the directory `$tntsymmetry` (`TNT_SYMMETRY:` on VMS). The name of the file is simply the name of the space group with a "`.dat`" appended. The file which includes the definitions of the space group $P4_322$ is `$tntsymmetry/p4322.dat`.

While the LATTICE statement recognized R centered cells, TNT's FFT code will not perform Fourier transforms in this setting. Rhombohedral cells must be described in their hexagonal setting.

Code	Lattice
P	Primitive
A	A Centered
B	B Centered
C	C Centered
I	Body Centered
R	Rhombohedral Centered
F	Face Centered

Table 3.1: Coded values for each lattice class

`LINK <Residue name>  <Target name>  <Linkage type>`

The link statement defines a stereochemical link between a residue in "the chain with no name" and a residue in a symmetry image of the chain. For instance, "LINK 128 128 SS" means that residue 128 is connected to a symmetry image of itself by a disulfide bond.

Additional information can be found in the "TNT Sequence File" chapter of the TNT Users' Guide.

`OPERATOR  <Symmetry operator> [; <Operator name>]`

The OPERATOR statement allows the input of the symmetry operators of the space group. One OPERATOR statement must be given for each symmetry operation, and the symmetry operator is given in the form used in the International Tables. Each operator may be given a name by the user. *Because the crystal lattice is given on the LATTICE statement you should not enter the centering operators on OPERATOR statements.* Only list the equivalent positions as listed in the International Tables.

To specify a name for an operator terminate the algebraic definition with a semicolon and enter the name at the end of the line. If the user does not do this the program will select a name for the operator. Here are some examples:

```
OPERATOR  X, Y, Z;      IDENT
OPERATOR  -Y, X-Y, Z;   MON1
OPERATOR  -X+Y, -X, Z;  MON2
```

The proper LATTICE and OPERATOR statements for all useful space groups are given in the directory `$tntsymmetry` (or `TNT_SYMMETRY:` on VMS). The name of the file is simply the name of the space group with a "`.dat`" appended. The file which includes the definitions of the space group $P4_322$ is `$tntsymmetry/p4322.dat`.

```
OPTION {SET | CLEAR} N(<Option name>)
```

This statement either sets or clears the options named. Different programs have different options which can be turned on or off by this mechanism. By default all options are cleared when the program starts.

The only option recognized by all programs is VERBOSE. When this option is SET the program will write detailed information to the standard output.

```
RESIDUE <Residue designator> <Residue type> -
                                  N(<Residue linkage>)

<Residue linkage> :== <Residue name> <Linkage type>
```

This statement defines the residue type of a particular residue within a chain type along with the targets and types of any linkages to other residues within the same chain. For a more detailed description of this statement the "TNT Sequence File" chapter of the TNT Users' Guide.

```
RESOLUTION <Inner limit> <Outer limit>
RESOLUTION <Outer limit>
```

This statement specifies the inner and outer resolution limits of the crystallographic data in Ångstroms. The order of the resolution limits on the statement is unimportant. If the inner limit is not given it is presumed to be equal to infinity, which means that all data from F(000) to the outer limit will be used.

```
TRANSFORMATION <Name> -
  {[{EULERANGLES 3<Values> TYPE {CROWTHER | HUBER | ROSSMANN} |
     POLARANGLES 3<Values> TYPE {KABSCH | ROSSMANN} |
     MATRIX 9<Values> |
     AXIS <Angle> 3<Direction Cosines> }]
   [CENTER 3<Values>]
   [SHIFT {XYZ 3<Values> | B <Value> | OCC <Value>}]
    SYSTEM {TNT | PDB | FRACTIONAL | GRID}
  |
   OPERATOR <Equivalent Position>}
```

Each transformation contains a rotation, a center, a translation, and a coordinate system. When the transformation is defined with the OPERATOR modifier all of them are defined at once. With the other modifiers the parts are defined separately. Each part which is undefined is assumed to be 1) a zero degree rotation, 2) centered at the origin, 3) no translation. In an attempt to avoid confusion the angular TYPE and coordinate SYSTEM cannot be defaulted.

In all cases the order of application is 1) the rotation about the center and 2) the translation.

A transformation with a particular name can only be defined once. If the determinate of the MATRIX is not equal to one a warning will be generated but the transformation will be accepted even though this will usually be mistake.

The Crowther Euler angles are defined as a rotation about the Z axis, followed by a rotation about the new Y axis, and finally a rotation about the ultimate Z axis. The form of the rotation matrix is $\begin{pmatrix} + & + & 0 \\ - & + & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

The Huber Euler angles are defined as a rotation about the Z axis, followed by a rotation about the new Y axis, and finally a rotation about the ultimate X axis. The form of the rotation matrix is $\begin{pmatrix} + & + & 0 \\ - & + & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

The Rossmann Euler angles are defined as Z, X, Z, with the form $\begin{pmatrix} + & + & 0 \\ - & + & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

In Kabsch polar angles the first two angles define the direction of an axis and the third angle defines a rotation about that axis. The first angle is the angle between the X axis and the projection of the rotation axis onto the XY plane. The second angle is the angle between the Z axis and the rotation axis. The first angle may have any value. The second angle must lie between 0 and 180 degrees. The third angle may also have any value.

In Rossmann polar angles the first two angles define the direction of an axis and the third angle defines a rotation about that axis. The first angle is the angle between the X axis and the projection of the rotation axis onto the XZ plane. The second angle is the angle between the Y axis and the rotation axis. The first angle may have any value. The second angle must lie between 0 and 180 degrees. The third angle may also have any value.

```
WEIGHT N(<Restraint class> <Value>)
```

WEIGHT statements are used to specify a weight for each module. Some modules, like that for stereochemistry, will accept a weight for each of several different classes of restraints, such as bond lengths and bond angles.

Standard TNT Command Statements

```
CHANGE {ATOM_TYPE | RESIDUE_TYPE | CHAIN_TYPE } -
       OF <Atom, Residue, or Chain name>  -
       [FROM <Current name>] TO <New name>
```

This statement is used to change the type of an atom, residue, or chain to some other type. It is most useful when converting file format. One can change the names used in the imported file to match the TNT standards automatically.

For example, there is a general problem with reading a PDB coordinate file in that the elemental type of each atom does not include the ionization state. That is, the file will indicate that a particular atom is copper but it does not say whether the atom has a +1 or +2 charge. The ionization state affects the shape of the atom and must be known to correctly calculate the scattering of the X-rays. The PDB file will contain the atom type field CU but TNT requires either CU+ or CU++. To correct this problem after the PDB file has been read enter the statement

CHANGE ATOM_TYPE OF *|COPPER:* FROM CU TO CU++

You will note that this particular example only changes the atom type of the atoms in residue COPPER. All other atoms of type CU will be unaffected. This behavior is what distinguishes the CHANGE command from the RENAME command.

```
INCLUDE <File name>
```

This command tells the program to read input from the file specified until its end is reached. Then the program continues to read from the statement after the INCLUDE. INCLUDEd files may contain INCLUDEs. File names can be no longer than 128 characters. The filename is passed to the operating system without being modified.

Short Name	Full Name
MAP	Ten Eyck Map Format
DSN6	Frodo DSN6 Map Format
HKL	Formatted Coefficients
PACKED	Unformatted Coefficients

Table 3.2: Possible crystallographic data file formats

```
PUNCH <Filename> {MAP | DSN6 | HKL | PACKED | ATOMx} -

            [BLUR <Value>] [SCALE <Value>] -
            [{GRID 3(<value>) | OVERSAMPLE <value>}] -
            [{LAYOUT 6(<value>) | ASYMMETRIC}]
```

The PUNCH command is used to write information to a file on disk. All programs can write their coordinate list in either ATOM, ATOMC, or ATOMG format. The program Convert can write coordinates in other formats as well.

Those programs which work with maps and Fourier coefficients can write these data in a number of formats. One may PUNCH map or coefficient data in the formats listed in Table 3.2. It usually does not matter to the program whether you choose to write this type of data as a map or a coefficient file. The program will perform the necessary calculations to fulfill your request. Please remember that a Fourier coefficient file is much smaller and quicker to write than a map file. Try to use coefficients files whenever possible.

Different programs will produce different kinds of maps or coefficients. Please refer to the individual program descriptions to discover what additional information is required on the PUNCH command to control the operation of the program.

The characteristics of the map or coefficient file can be affected by the SCALE, or BLUR modifiers. Further a map file can be modified by the GRID, OVERSAMPLE, LAYOUT, and ASYMMETRIC modifiers.

The SCALE modifier is used to multiply all points in the map or all the Fourier coefficients by a constant factor. The BLUR modifier is used to add a temperature factor. (A blur cannot be applied to a map read from disk. The calculations are simply too costly.)

If a coordinate file has been given, the default map layout is a molecular volume. If no coordinate file has been supplied or the ASYMMETRIC modifier has been included the map file will contain an asymmetric unit.

The program will choose a sampling rate (or grid) suitable for the kind of data it is writing to the map file. The user may override the choice of the program with the OVERSAMPLE or GRID modifier. OVERSAMPLE is used to specify the sampling rate as a fraction of the outer resolution limit. An OVERSAMPLE of three means to sample the map three times for each distance equal to the resolution limit. The sampling rate can be defined explicitly with the GRID modifier. When using GRID the user must be careful to obey the (complicated) rules which limit the values which can be chosen. These restrictions are discussed in Appendix D.

One can generate a map with an arbitrary layout with the LAYOUT modifier. The six numbers following LAYOUT exactly define the limits of the map to be written. If LAYOUT is used the grid must be defined on the same line with GRID. The six values following the LAYOUT modifier are defined to be (xmin, xmax, ymin, ymax, zmin, zmax) in the grid units.

A Fourier coefficient file always contains an asymmetric unit of data.

```
RENAME {ATOM <Chain>|<Residue>:<Atom> | ATOM_TYPE <Atom type> |
        RESIDUE <Chain>|<Residue> |
        RESIDUE_TYPE <Residue type> |
        CHAIN <Chain> | CHAIN_TYPE <Chain type>} -

       [FROM <Current name>] TO <New name>
```

This statement is used to change the name of some item with which the program is currently familar. One can change the name of a particular atom, residue, or chain. In addition, the name of an atomic type, residue type, or chain type can be changed. It is most useful when converting file format. One can change the names used in the imported file to match the TNT standards automatically.

For example, there is a general problem with reading PDB format coordinate files in that the elemental type of each atom does not include the ionization state. That is, the file will indicate that a particular atom is copper but it does not say whether the atom has a +1 or +2 charge. The ionization state affects the shape of the atom and must be know to correctly calculate the scattering of the X-rays. The PDB file will contain the atom type field CU but TNT requires either CU+ or CU++. To correct this problem after the PDB file has been read enter the statement

```
RENAME   ATOM_TYPE   CU   TO CU++
```

You will note that this statement changes all references to the old atom type CU to CU++, you cannot change the type of only one atom out of a group with the RENAME command.

As a further example consider the problem posed by an old method for naming the atoms in the side chains of amino acids. In this system the terminal nitrogen atoms of an arginine residue were named NEE1 and NEE2 instead of the current form NH1 and NH2. When reading a file from these bygone days these names must be changed. This operation is accomplished with the commands

```
RENAME   ATOM *|*:*   FROM NEE1   TO NH1
RENAME   ATOM *|*:*   FROM NEE2   TO NH2
```

Note that by using wild cards in specifying the atoms to be renamed we avoid having to figure out which residues are arginines.

Chapter 4

The Stereochemical Module

This module analyzes the agreement between the present model's stereochemistry and ideal geometry. The precise functional form of this term is the sum of the square of the deviation from ideality for every bond length, bond angle, torsion angle, pseudorotation angle, trigonal carbon, general plane, and temperature factor correlation. It also adds in the sum of the square of the difference between the minimum distance allowed between two non-bonded atoms and the distance actually found whenever the two atoms are too close together.

A feature of this module is that any geometry restraint can span between a molecule in the asymmetric unit and one in a neighboring asymmetric unit. This ability is useful for keeping solvent molecules on the surface in line, and restraining the geometry of cross-links, or larger solvent molecules which sit on special positions.

All of the functions of this module are completed with the program Geometry. To perform the calculations required for a long loop one would use either the CURVATURE or GRADIENT command. In a short loop the FUNCTION command should be used because it causes only the function value to be calculated.

GEOMETRY

Stereochemical Program

This program takes a coordinate file and a definition of the ideal geometry for that structure and calculates the function value and the first and second derivatives with respect to the positional parameters of the atoms. It can also provide various statistical measures of how closely the molecular model obeys the ideal geometry restraints. The stereochemical restraints this program understands are bond lengths, bond angles, torsion angles, pseudorotation angle of five membered rings, planarity about a trigonal atom, general planarity, non-bonded contacts, B correlation, and chiral centers. Chirality is a two state, discontinuous function and therefore cannot be refined. The program informs the user when an atom with incorrect chirality is encountered.

The program has functions beyond that of refinement. With the SCREEN command one can determine the places in the model with the greatest disagreement with standard geometry. The REPORT GEOMETRY command can be used to compare the average and standard deviation for each geometry restraint, as determined by actually looking at coordinates, with any particular library. This allows one to generate new libraries, or look for errors in the existing ones.

Required Input

Geometry deals with stereochemistry. It must know the present structural model of the molecule and the standard geometry for that molecule. The means of describing this information is presented in the "TNT Sequence File" and the "Standard Geometry Definition" chapters of the TNT Users' Guide. To allow the program to find "bad" contacts it must be given the minimal distances allowed between each atom type as well as the symmetry operators for the structure's space group. This information is entered on CONTACT, LATTICE, and OPERATOR statements.

Options

Geometry recognizes two options. These options can be set with the OPTION statement described in the "Standard TNT Data Statements" section on Page 27. As an example, the activation of the VERBOSE option would require the statement

```
OPTION SET VERBOSE
```

- NOCONTACTS

  This option causes the program to ignore all potential bad contacts in all its operations. This is useful in the early refinement of models generated by molecular replacement. One usually prefers to refine the model a bit before model building to fix up the crystal contacts.

- VERBOSE

  The Verbose option causes the program to write additional information to the log file about the choices it makes.

Unique Input Statements

```
CONTACT  <Type> <Type> <Distance> <Standard deviation>
```

The CONTACT statement defines the allowed contact distances for non-bonded contacts between atoms. Contacts between non-bonded atoms are screened every time the other geometry restraints are checked, and if two non-bonded atoms are closer than the minimum contact distance they will be flagged as a bad contact and will be pushed apart by refinement. Whether two atoms are bonded is determined by checking the list of standard bond lengths. All 1-2, 1-3, and 1-4 contacts are ignored. These kinds of interactions are covered in bond length, bond angle, and torsion angle refinement, respectively, and should not be considered again.

The information that should be placed on this statement is the elemental types of the two elements, the minimum contact distance, and the individual standard deviation for that particular contact, in the following form:

CONTACT O C 2.8 0.1

This means that the minimum non-bonded contact distance allowed between an atom of carbon and an atom of oxygen is 2.8Å with a sigma of 0.1Å. (Note: This restraint is based only on the type of the atoms involved where the other kinds of restraints depend on the name of the atom and the residue type.)

A library of CONTACT statements for many atom types is provided in the file `$tntdata/contact.dat` (in `TNT_DATA:CONTACT.DAT` on VMS). A description of the sources of information used to generate these values is contained within this file.

> It is quite difficult to find a proper set of values for the closest allowed distances between to element types. The values in `contact.dat` have been purposely chosen to be small. This ensures that a contact flagged "bad" will really

be bad. However there are some cases where the local chemistry allows two atoms to approach one another so closely the the library defines a contact as "bad" when it really is not. These particular cases should be EXCLUDEd.

```
EXCLUDE  [<Chain>|]<Residue>[:<Atom>] -
         [<Chain>|]<Residue>[:<Atom>]
```

This statement causes any close contacts between atoms in one group with any atom in the other group to be ignored when deciding what the bad contacts are. A group is defined by giving the chain name, residue name, and atom name of the atom whose contacts are to be ignored. Wild card may be placed in any position. The default residue and atom names are wild cards, but the default chain name is the "chain with no name". To specify an entire residue one gives the residue name. To specify a particular atom one gives the residue name and the atom name separated by a colon. To specify all atoms with the same name leave the residue name blank and provide a wild card ("*") as the chain name. Here are some examples.

```
EXCLUDE  143   167
EXCLUDE  67:O  SOL1:O65
EXCLUDE  *|:O  *|:C
```

In the first example all contacts between any atoms in residue 143 with atoms in residue 167 will be ignored. The second example states that if a bad contact is found to exist between atom O of residue 67 and atom O65 of residue SOL1 that contact will be ignored. In the third example the program will ignore all contacts between atoms named O and atoms named C anywhere in the structure.

Here is an example of a use of the EXCLUDE statement which uses the oligomer definition of Cro given as an example in the "TNT Sequence File" chapter of the TNT Users' Guide.

EXCLUDE O| O|
EXCLUDE A| A|
EXCLUDE B| B|
EXCLUDE C| C|

These EXCLUDE statements will cause all non-bonded contacts within CRO chains to be ignored leaving only interchain contacts to be listed and refined.

```
WEIGHT BOND  <x> TORSION  <x> PLANE   <x> PSEUDO  <x>
WEIGHT ANGLE <x> TRIGONAL <x> BCORREL <x> CONTACT <x>
```

This statement is used to specify the weights to be applied to each class of stereochemical restraint. Each name of a restraint type is followed by the weight that type is to be given. Chiral centers cannot be refined and therefore there is no way to define a weight for that class of restraints (If you want to call something that cannot be refined a restraint).

There is a great deal of flexibility in the arrangement of the weight definitions. The weights can be defined in any order, and can be placed on several different WEIGHT statements. Geometry will ignore weights from other modules.

Command Statements

These statements cause Geometry to perform a particular action using the data that has already been entered on other statements. Most of the examples INCLUDE the file `$tntdata/tntgeo_v010.dat`. This file contains the full set of standard geometry for TNT.

In addition your control file, presumed here to be named `your.tnt`, must include the CELL statement, space group definition, sequence file, and your weights. In addition, your control file must also defined the standard geometry for any unusual groups in your model.

```
CURVATURE <File name>
```

This command is used in the long loop of refinement. It causes the function value, and the first and second derivatives of the geometry restraints to be calculated and written to the file specified. This command must be preceded by at least the geometry definition, a coordinate set, the space group, and a WEIGHT statement.

```
$tntbin/geometry << $eof
INCLUDE $tntdata/tntgeo_v010.dat
INCLUDE your.tnt
INCLUDE init.cor
CURVATURE geometry.dat
$eof
if ($status >< 0) then exit 1
```

```
FUNCTION  <File name>
```

This command is used in the short loops of refinement. It causes only the function value for the geometry restraints to be calculated and written to the file specified. This command must be preceded by at least the geometry definition, a coordinate set, the space group, and a WEIGHT statement.

```
$tntbin/geometry << $eof
INCLUDE $tntdata/tntgeo_v010.dat
INCLUDE your.tnt
INCLUDE shifted.cor
FUNCTION  geometry.dat
$eof
if ($status >< 0) then exit 1
```

```
GRADIENT <File name>
```

This command is used in the long loop of refinement when the method of minimization is to be either steepest descent or conjugate gradient. It causes the function value and gradient for the geometric restraints to be calculated and written to the specified file. This command must be preceded by at least the geometry definition, a coordinate set, the space group, and a WEIGHT statement.

```
$tntbin/geometry << $eof
INCLUDE $tntdata/tntgeo_v010.dat
INCLUDE your.tnt
INCLUDE init.cor
GRADIENT geometry.dat
$eof
if ($status >< 0) then exit 1
```

`REPORT {GEOMETRY | STRUCTURE | SYMMETRY}`

There are three different report types listed below.

- GEOMETRY

  This command causes the printing of the geometry statistics on a per geometry restraint basis. For each restraint entered in the geometry library the mean and r.m.s. deviation from the standard value of the occurrences of that restraint will be listed. For example, this listing would contain the average and r.m.s. deviation for the CA, CB bond length in all ALA residues in the structure. This listing is useful when looking for systematic deviations from the standard value of a geometry restraint in the library.

- STRUCTURE

  This command causes the printing of a list of a small number of interesting things about a structure. If a CELL statement has been given the cell constants will be echoed along with the reciprocal cell constants and the orthogonalization and deorthogonalization matrixes. If atomic coordinates have been given the number of atoms will be listed along with the number of residues in the structure. In addition the location and extent of each chain will be printed.

- SYMMETRY

  This command causes the printing of a list of the symmetry operators that have been entered. Each symmetry matrix is printed in crystallographic coordinates with the translation vectors added so that when a symmetry operator is applied to a molecule a nearest neighbor molecule is generated. Each operator is listed with the name that the SCREEN listing uses to refer to that operator when a bad contact is found between atoms of two symmetry related molecules.

```
SCREEN <Number>
```

This command is used to get a list of the worst so many bonds, angles, etc.. It requires that the geometry library, space group, and a coordinate set be given.

```
$tntbin/geometry << $eof
INCLUDE $tntdata/tntgeo_v010.dat
INCLUDE your.tnt
INCLUDE shifted.cor
SCREEN  20
$eof
if ($status >< 0) then exit 1
```

Chapter 5

The Noncrystallographic Symmetry Module

The function minimize by this module, when the noncrystallographic symmetry is used as restraints is

$$\sum_{c=1}^{\text{clusters}} \sum_{i=1}^{\text{chains}-1} \sum_{j=i+1}^{\text{chains}} \sum_{atm=1}^{\text{atoms}} |(\mathbf{R}_{ci}\mathbf{x}_i(atm) + \mathbf{t}_{ci}) - (\mathbf{R}_{cj}\mathbf{x}_j(atm) + \mathbf{t}_{cj})|^2$$

Thus, for each cluster there is a loop over all unique pairs of chains. The distance between each corresponding atom, after being moved to the prototypical location, is squared and summed.

A basic assumption of this module is that the objects which are related by the noncrystallographic symmetry are identified as individual chains, where the equivalent residues are given the same name in each chain. The old style of simply giving the second molecule's residues numbers much larger than the first will not work here. This assumption also means that the related molecules must be chemically identical. You cannot define noncrystallographic relations between two chains which are only similar in sequence, they must be identical.

When refining a model in the presence of noncrystallographic symmetry constraints the parameters of the model can be stored in the coordinate file in two states. In one state the model is expanded by the symmetry and the full asymmetric unit is present. In the other state only the unique atoms of the model are present. The nomenclature for these two states are the “scattered” state and the “gathered” state. The **scatter** and **gather** commands are used to change from one state to the other.

In the scattered state the atoms are named in the usual fashion. On the other hand, in the gathered state the atoms related by ncs symmetry are placed in a pseudo-chain given the name from the relevant CLUSTER statement. Because of this change in chain name a gathered coordinate set cannot be used in crystallographic calculations without being scattered first.

NCS

Noncrystallographic Symmetry Program

NCS is a program for performing the basic operations required to use noncrystallographic symmetry in refinement. It has two classes of commands depending on whether this symmetry is being constrained or restrained. When noncrystallographic symmetry constraints are being applied the SCATTER, GATHER, and PUNCH commands are used. With restraints the CURVATURE, GRADIENT, and FUNCTION commands are used. These commands are described in the "Command Statements" section.

Unique Input Statements

```
CLUSTER <Name> N([RESIDUES <Residue range>]) -
               {CHAINS N(<Chain>) |
               NCS N(<Chain> <Operator>)}
```

The CLUSTER statement is used to define the chains which are related by noncrystallographic symmetry, and the portions of those chains for which the correspondence is valid. At least one CLUSTER statement must be given each time NCS is ran. If a residue range is given the sequence file must be INCLUDEd.

The CHAINS options will continue reading chain names until the end of line is reached. Therefore all RESIDUES options must come before it.

The NCS option is used to define the name of the operator which moves that chain onto the prototype. The actual operators are supplied on TRANSFORMATION statements. The NCS option will continue to read pairs of chain names and operator names until the end of line is reached. Therefore CHAINS and NCS must be entered on separate CLUSTER statements.

```
WEIGHT NCS <Value>
```

This statement is used to specify the weight of the noncrystallographic symmetry in relation to the other types of restraints currently being used. The weight should be chosen so that the resulting agreement between related molecules matches your expectations. Large molecules with good noncrystallographic symmetry should agree with an r.m.s. comparable to the error in the coordinates. The actual r.m.s. may be higher if there are real differences between the mates.

There is a great deal of flexibility in the arrangement of the weight definitions. The weights can be defined in any order, and can be placed on several different WEIGHT statements. NCS will ignore weights from other modules.

Command Statements

Here follows a description of the command statements in NCS. Since is would be pointless to run this program if there were no noncrystallographic symmetry the definition of the regions of the model covered by this symmetry is required for all commands. These definitions are made with the CLUSTER statements defined above. The examples given for each command assume that the CLUSTER statements have been placed in the TNT control file `yours.tnt`. The examples given below presume that the NCS weight and sequence file is also given in the control file.

The SCATTER command requires, in addition, the noncrystallographic symmetry operators, since they cannot be calculated from the unique atoms.

```
CURVATURE <File name>
```

This command is used in the long loop of refinement. It causes the function value, and the first and second derivatives of the NCS restraints to be calculated and written to the file specified. This command must be preceded by at least the NCS definition, a coordinate set containing a full set of chains, and a WEIGHT statement. You will need the sequence file if residue ranges are used in the cluster definition.

The noncrystallographic symmetry operators themselves are optional. If they are not supplied NCS will use the current coordinates to derive them.

```
$tntbin/ncs << $eof
INCLUDE yours.tnt
INCLUDE init.cor
CURVATURE ncs.dat
$eof
if ($status >< 0) then exit 1
```

`FUNCTION  <File name>`

This command is used in the short loops of refinement. It causes only the function value for the NCS restraints to be calculated and written to the file specified. This command must be preceded by at least the NCS definition, a coordinate set containing a full set of chains, and a WEIGHT statement. You will need the sequence file if residue ranges are used in the cluster definition.

The noncrystallographic symmetry operators themselves are optional. If they are not supplied NCS will use the current coordinates to derive them.

```
$tntbin/ncs << $eof
INCLUDE yours.tnt
INCLUDE shifted.cor
FUNCTION  ncs.dat
$eof
if ($status >< 0) then exit 1
```

`GATHER`

The GATHER command will average together all the symmetry mates in a noncrystallographic cluster to produce a prototype molecule for each CLUSTER statement. It averages the coordinates, gradients, and curvatures for each atom in each cluster. The unaveraged atoms are then removed from the atom list in the program.

The GATHER command is used when refining a model with noncrystallographic constraints. for example, after Rfactor has calculated the gradient and curvature for the entire cluster (or set of clusters) NCS will move those average values to the prototype's orientation and average them. The averaged gradient and curvature is presented to Shift instead of the originals.

The noncrystallographic symmetry operators themselves are optional. If they are not supplied NCS will use the current coordinates to derive them. Either the operators must be supplied on TRANSFORMATION statements or the coordinates must be given. If both are supplied the TRANSFORMATION statements have precedence.

In the example below, the coordinates from `init_expanded.cor` and the gradient and curvature from `rfactor_expanded.dat` are gathered into the CRO prototype using the noncrystallographic symmetry operators from the file `cluster.ncs` After the gathering the results are PUNCHed to `rfactor.dat`

The TNT control file is presumed to contain the CLUSTER statement "CLUSTER CRO CHAINS O A B C". The cluster is named CRO on the CLUSTER statement. `init_expanded.cor` contains four chains — O, A, B, and C. `rfactor.dat` contains only one — CRO.

```
$tntbin/ncs << $eof
INCLUDE cro.tnt
INCLUDE init_expanded.cor
INCLUDE rfactor_expanded.cor

INCLUDE cluster.ncs
GATHER
PUNCH rfactor.dat CURVATURE
$eof
if ($status >< 0) then exit 1
```

```
GRADIENT <File name>
```

This command is used in the long loop of refinement when the method of minimization is to be either steepest descent or conjugate gradient. It causes the function value and gradient for the noncrystallographic symmetry restraints to be calculated and written to the file specified. This command must be preceded by at least the NCS definition, a coordinate set containing a full set of chains, and a WEIGHT statement. You will need the sequence file if residue ranges are used in the cluster definition.

The noncrystallographic symmetry operators themselves are optional. If they are not supplied NCS will use the current coordinates to derive them. If both are given the operators on the TRANSFORMATION statements are used.

```
$tntbin/ncs << $eof
INCLUDE cro.tnt
INCLUDE init.cor
GRADIENT ncs.dat
$eof
if ($status >< 0) then exit 1
```

```
PUNCH <File name> {ATOM | ATOMC | ATOMG |
                   GRADIENT | CURVATURE |
                   NCS}
```

The PUNCH command is used to get information out of NCS. The actual information produced depends on the format of the file requested.

- ATOM, ATOMC, ATOMG

  The program writes a coordinate file with the name and format designated. The command must be preceded by a coordinate set. If ATOM format is used a CELL statement should be given and, in addition, a GRID statement should be given if ATOMG is used.

- GRADIENT, CURVATURE

  The program writes the current gradient (and curvature if CURVATURE was specified) to the named file. When applying noncrystallographic symmetry constraints, this command is used after the GATHER command has averaged the gradient and curvature from each individual chain into that of the prototype.

- NCS

  The program writes the full list of noncrystallographic symmetry operators to the file using CLUSTER and TRANSFORMATION statements. If a set of operators were not entered into the program the current coordinates and CLUSTER statements will be used to determine these operators.

`SCATTER`

This command is used to expand the prototype molecule by the noncrystallographic symmetry operators to generate the entire cluster of molecules. It will expand all occurrences of noncrystallographic symmetry. All of the current information about the atoms of the prototype are expanded: coordinates, gradients, and curvatures. The prototype, itself, will be removed from the list of coordinates.

The noncrystallographic symmetry operators must be supplied to use the SCATTER command, in this example in the file `ncs_symmetry-.dat`. In addition you will need the sequence file if residue ranges are used in the cluster definition.

```
$tntbin/ncs << $eof
INCLUDE cro.tnt
INCLUDE shifted.cor
INCLUDE ncs_symmetry.dat
SCATTER
PUNCH shifted_all.cor ATOMC
$eof
if ($status >< 0) then exit 1
```

`SCREEN <Number>`

This command is used to get a list of the worst agreements between the similar chains. It will only consider the regions of similarity defined on the CLUSTER statements. It requires that the NCS definition and a coordinate set be given. You will need the sequence file if residue ranges are used in the cluster definition.

Of course, if you are constraining ncs there will be no deviants to list.

```
$tntbin/ncs << $eof
INCLUDE cro.tnt
INCLUDE shifted.cor
SCREEN  20
$eof
if ($status >< 0) then exit 1
```

Program Operation

When refining a model using noncrystallographic symmetry restraints the program NCS is run is a fashion similar to the modules for other types of observations. It is given the current coordinate set and a definition of the noncrystallographic symmetry. In the long loop the CURVATURE command is used while in the short loop it is the FUNCTION command.

When using constraints the program is run in a very different fashion. It now becomes a pre- and postprocessor for the other modules. The atoms of the prototype must be expanded to fill the asymmetric unit of the crystal. After the modules have completed, NCS is run to reduce the gradient and curvature back to the prototype. In the short loops the the function values do need to be reduced so NCS only needs to be run to SCATTER each `shifted.cor`.

One cannot construct a generic script for refining a model with constrained noncrystallographic symmetry. The problem is that the model must be "scattered" when the modules are ran but "gathered" when Shift is ran. This means that the all references to atoms in the control file must be made to atoms in the individual chains but the COMBINE and CONSTANT statements must refer to the prototypical chains. They cannot be placed in the control file. A script must be edited to place the proper parameter editing statements in the input for Shift.

Here is an example of a script which constrains a model to noncrystallographic symmetry. This file is kept in `$refroot/doc/prototype/-tnt_cycle_con_ncs`. As usual the coordinates are stored in `init.cor` and the final gradient and curvature are stored in `rfactor.dat` and `geometry.dat`. The CLUSTER statement is given in the control file. The transformations which relate the individual chains to the prototype are stored on CLUSTER and TRANSFORMATION statements in a file by themselves. The prototype script needs to be modified at the places marked by "<<< ... >>>".

Chapter 6

The X-ray Crystallographic Module

This module calculates the function value, gradient, and curvature for the term of the function that involves the crystallographic data. This term is

$$f(\mathbf{x}) = \sum_{hkl} \frac{1}{\sigma_o^2(hkl)} (|F_o(hkl)| - |F_c(hkl, \mathbf{x})|)^2.$$

Because of the number of reflections in the average protein structure and the amount of time it takes to calculate them from a structural model, this module uses the largest portion of CPU time of the entire refinement cycle.

In a long loop this module must be able to calculate the function value, gradient, and curvature of the function. To calculate any of these quantities structure factors must be calculated from the current model and scaled to the observed structure factors. Once this is done the function value and curvature can be directly calculated. The gradient is calculated via the method of Agarwal which requires that a difference map be convoluted with a separate function for each parameter in the model. The program Rfactor performs all these tasks.

In the long loop both the term's value and gradient must be calculated while in a short loop only the term's value is required. The curvature will be required if the gradient/curvature or preconditioned conjugate gradient methods of refinement are to be used.

RFACTOR

Structure Factor Comparison Program

Rfactor handles many of the functions that are required when two lists of structure factors are compared. It examines two sets of structure factors and finds all reflections which occur in both. It will scale the two data sets to one another, allowing the user to choose between many options. It can calculate the R-value between the two lists, and also generate difference Fourier coefficients and maps of several types. In addition it will generate the function value, gradient, and curvature for the RFACTOR, REALSPACE, ROTATION, and PHASE modules. In refinement it is run in both long and short loops. It can either read the Fc's from a file or calculate them from a map or coordinates.

In the long loop of a refinement cycle Rfactor compares the Fo's and Fc's and determines the parameters of the scaling function. The program also writes out the parameters of the scale function so that they may be read back during the following short loops. It then calculates and writes the function value and gradient for this term. If the curvature is desired RFACTOR will calculate it as well.

In a short loop the parameters of the scaling function will be read from the file produced in the long loop and the crystallographic term's sum will be calculated using those scale factors.

Rfactor calculates the maps used when model building. It can produce Fo-Fc, 2Fo-Fc, 3Fo-2Fc, and Fo-Fo maps in either Ten Eyck MAP format or A. Jones' DSN6 format.

The Scaling Function

The scaling function is constructed so that it can compensate for anisotropic decay in the scattering and the lack of a solvent continuum in the model. (The solvent continuum correction was suggested in Moews and Kretsinger (J. Mol. Biol. (1975). **91**, 201-228)). The function is

$$\exp -\frac{1}{4}Bs^2 \exp -\frac{1}{4}\mathbf{s}^t\mathbf{B}_{aniso}\mathbf{s} \left(1 - K_{sol}\exp -\frac{1}{4}B_{sol}s^2\right),$$

$$\text{where } \mathbf{s} = \begin{pmatrix} h & k & l \end{pmatrix} \text{ and } s = \sqrt{\mathbf{s}^t\mathbf{s}}.$$

The parameter B is an isotropic temperature factor correction. $\mathbf{B}_{aniso}$ is a purely anisotropic temperature factor correction. K_{sol} is the ratio of the average solvent density and the average protein density. B_{sol} is an additional temperature factor used to blur the solvent relative to the protein. In addition there is a scale factor (K), independent of resolution, by which Fo is multiplied, which brings the observations onto an absolute scale. You have the option of either allowing Rfactor to choose a value for each of these parameters, or of setting any or all of the values yourself.

The anisotropic temperature factor ($\mathbf{B}_{aniso}$) is a symmetric matrix containing 6 unique elements. They are named B11, B22, B33, B12, B13, and B23. Many space groups will restrict the values of these elements. Rfactor takes care of enforcing these restrictions. The matrix is required to be purely anisotropic and this is ensured by the requirement that B11, B22, and B33 sum to zero.

Using the SET Statement

There are three general cases where Rfactor is run: calculating an R-value, calculating a map or Fourier coefficients, and running refinement.

When calculating an R-value between two data sets, or producing a map or coefficients, you want the best scale factors so that you can get the best agreement between the two data sets. Therefore, you would allow Rfactor to choose all scale factors.

During refinement you want the individual temperature factors of the model to absorb the signal which causes the parameter B to be non-zero. The model cannot adjust to this signal if the signal has already been eliminated. B must be fixed with a value of zero in all refinement, and Rfactor does this automatically.

Additional restrictions must be placed on the scale factors if the data sets do not cover large areas of reciprocal space. These matters are discussed in the "Setting Up a Project" chapter of the TNT Users' Guide.

Options

Rfactor recognizes four options. These options can be set with the OPTION statement described in the "Standard TNT Data Statements" section of the "Shared I/O Properties" chapter. The first two options simply affect how the agreement statistics are printed in the log file. The SIGMA option makes a substantive change to the refinement method. As an example, the activation of the SIGMA option would require the statement

OPTION SET SIGMA

- FOM

  This option causes the Rfactor command "REPORT RFACTOR" to print additional statistics describing the agreement between the two data sets. These statistics are weighted by the figure of merit and show the means and RMS's when those reflections which are well phased are given greater weight.

- INTENSITY

  This option causes the Rfactor command "REPORT RFACTOR" to list the statistics in intensity shells instead of the usual resolution shells.

- SIGMA

  This option causes Rfactor to weight each reflection by $1/\sigma_o^2$ when calculating the function value, curvature, and Fourier coefficients. The SIGMA option is preferred if the sigmas in the FO file are reliable. When this option is set additional statistics are printed in the log file. Sigma weighting is used when calculating scale factors, anisotropic temperature factors, Sim weighting coefficients, the RFACTOR function value and curvature, the ROTATION function value and curvature, and difference coefficients.

- VERBOSE

  The Verbose option causes the program to write additional information to the log file about the choices it makes.

Unique Input Statements

```
ACCEPT FO-FC <Upper limit>
ACCEPT FO    <Lower limit> <Upper limit>
ACCEPT FC    <Lower limit> <Upper limit>
```

This statement allows the user to specify particular limits to the acceptable values for the diffraction data. It permits limits to be placed on the value of FO, FC, and FO-FC. These limits are specified in the units of the diffraction data after scaling. Therefore all data is used to determined the scale factors.

```
FC  <File name> [FORMAT {HKL | PACKED | MAP}]
```

This statement gives the name of the file which is the source of the calculated structure factors. If a map file is given the calculated structure factors are arrived at by inverting the map. If no FC statement is given the Fc's will be calculated from the coordinates of the model.

```
FILE <File Id> <File name> [FORMAT (HKL | MAP | PACKED)] -
                           [SCALE <Value>] [BLUR <Value>]
```

This command describes an external binary data file to the program. This file is subsequently referred to by the arbitrary "File Id". If the file's format is not HKL, which is the default, the type of file may be given via the FORMAT option. When the data are read from the file a scale factor and blur may be applied. Because of the extreme difficulty of applying a blur to an electron density map, when the file is a map file, the BLUR option is presumed to indicate that the map has a already been blurred by the indicated amount.

In Rfactor this statement is only useful when using the AGARWAL and NORMAL_MATRIX commands.

```
FO  <File name> [FORMAT {HKL | PACKED | MAP}]
```

This statement gives the name of the file which is the source of the observed structure factors. If a map file is given the observed structure factors are arrived at by inverting the map. If no FO statement is given the Fo's will be calculated from the coordinates of the model. (I know this sounds a little weird, to define the structure factors calculated from a model to be Fo's. However sometimes it is useful to compare two calculated data sets, such as when someone wishes to determine just how different two models are.)

```
SET [K <Value>] [B <Value>] [KSOL <Value>] [BSOL <Value>]
    [B11 <Value>] [B22 <Value>] [B33 <Value>]
    [B13 <Value>] [B13 <Value>] [B23 <Value>]
```

This statement allows the user to set the value of any of the scaling function parameters. Any parameters which are not explicitly set are calculated by Rfactor by minimizing the squared difference between the two data sets. A SET statement clears all information about previous SET statements and resets any scale factors which have already been determined.

If any anisotropic scale factor component is set to zero they all are. If K_{sol} is set to zero B_{sol} is as well.

If the SIGMA option is set, any unspecified parameters are fit with each reflection weighted by $1/\sigma_o^2$.

```
SIM [<A> <B>]
```

This statement causes Sim weighting to be applied to any difference coefficients later produced with the PUNCH command. This statement would be used whenever a Sim weighted map is desired. The two parameters are optional. If they are given (As with all programs in this package typing 0.0 is different than typing nothing at all. 0.0 is accepted as a zero, not as an indication that the default is desired.) they specify a single Gaussian fit to the r.m.s. difference coefficients. If they are not given on the statement the program will calculate the best values for A and B and apply them to the data.

If the SIGMA option is set the data used to determine A and B are weighted by $1/\sigma_o^2$.

```
WEIGHT [RFACTOR <Value>] [ROTATION <Value>]
WEIGHT [PHASE <Value>]   [REALSPACE <Value>]
```

This statement allows the user to specify a weighting factor that will be applied to the function value, gradient, and curvature written out by the FUNCTION, GRADIENT, and CURVATURE commands. If a weight for the current module is not given a weight of 1.0 is assumed. During the short loops the same weight *must* also be applied. To ensure that this will occur you should keep the WEIGHT statement in your TNT control file.. If a weight for another restraint (such as BOND, or ANGLE) is given to this program on a WEIGHT statement the additional weights will be ignored.

Command Statements

In the following section each of Rfactor's commands are described and a small example is given. In these examples many of the required parameters are presumed to be read from the TNT control file, here assumed to be named `your.tnt`. Rfactor presumes that this file will contain the cell constants, the definition of the space group, the name of the observed structure factor file, and the resolution limits of the data to be considered. Some commands do not need all of this information but you will want it in your control file anyway.

```
AGARWAL <File name>  SOURCE <File Id> -
               [MODULE <Name>] -
               [BLUR <Value>] -
               [{GRID 3(<Value>) | OVERSAMPLE <Value>}]
```

The AGARWAL command causes the program to calculate the gradient of a function via an Agarwal-type convolution. The Fourier coefficients described with the SOURCE modifier will be used to generate the "difference map" required by the calculation. The <File Id> must have been defined on a FILE statement. The resulting gradient is written to the file <File name> with whatever module identifier was specified. You may override the default choice for blur and grid using the proper modifiers. By default the blur and grid will be chosen by Rfactor to provide accurate results in a short amount of time.

A detailed description of the Fourier coefficients which you are required to calculate can be found in the "Creating Your Own Module" chapter.

```
$tntbin/rfactor << $eof
INCLUDE $tntdata/formfactor.dat
INCLUDE your.tnt
INCLUDE init.cor
FILE COEFS  fo-fc.pak FORMAT PACKED
AGARWAL rfactor.dat SOURCE COEFS MODULE MY_MODULE
$eof
if ($status >< 0) then exit 1
```

```
CURVATURE <File name> -
         [MODULE {RFACTOR | REALSPACE | PHASE}]
```

This statement causes the function value, gradient, and curvature for the specified module to be calculated and written to the file name entered on the input statement. (The default module is RFACTOR.) The scale factor required in the short loop are written to the file `scales.dat`.

The CURVATURE command cannot be used with the ROTATION module. The calculation of curvatures and gradients in this module are more complicated than the others. You need to read the "Rotation Function Refinement Module" section of the "Other Modules in TNT" chapter.

Required for these calculations are the cell constants, the resolution limits, the symmetry operators, the lattice class, the Fo file name, the atomic form factors, the coordinate file, and the weight for this term. An example follows:

```
$tntbin/rfactor <<$eof
INCLUDE $tntdata/formfactor.dat
INCLUDE your.tnt
INCLUDE init.cor
CURVATURE rfactor.dat
$eof
if ($status >< 0) then exit 1
```

```
FUNCTION <File name> -
         [MODULE {RFACTOR | REALSPACE | PHASE | ROTATION}]
```

This statement causes the observed and calculated structure factors to be compared, and the proper scale factors that maximize the agreement between the two data sets to be calculated (after consideration of the most recent SET statement). Then the function value for the specified module will be determined. This function value will be written in the file whose name appears on the statement. An example where this statement is used in a short loop in refinement follows:

```
$tntbin/rfactor <<$eof
INCLUDE $tntdata/formfactor.dat
INCLUDE your.tnt
INCLUDE scales.dat
INCLUDE shifted.cor
FUNCTION rfactor.dat
$eof
if ($status >< 0) then exit 1
```

The file `scales.dat` contains the SET statement which reflects the scale factors used in the long loop. It is written whenever a GRADIENT or CURVATURE command is executed.

```
GRADIENT <File name> -
         [MODULE {RFACTOR | REALSPACE | PHASE}]
```

This statement causes the function value and gradient for the specified module to be calculated and written to the file name on the input statement. The scale factors needed in the short loops are written to the file `scales.dat`.

The GRADIENT command cannot be used with the ROTATION module. The calculation of gradients in this module is more complicated than the others. You need to read the "Rotation Function Refinement Module" section of the "Other Modules in TNT" chapter.

Required for these calculations are the cell constants, the resolution limits, space group symmetry, the Fo file name, the coordinates of the model, the formfactors, and the weight for this term. An example follows:

```
$tntbin/rfactor <<$eof
INCLUDE $tntdata/formfactor.dat
INCLUDE your.tnt
INCLUDE init.cor
GRADIENT rfactor.dat
$eof
if ($status >< 0) then exit 1
```

```
NORMAL_MATRIX <File name> SOURCE <File Id> -
                [MODULE <Name>] -
                [BLUR <Value>] -
                [{GRID 3(<Value>) | OVERSAMPLE <Value>}]
```

The NORMAL_MATRIX command allows one to calculate the diagonal blocks (one block per atom) of the normal matrix for any kind of crystallographic residual function. Using the procedure outlined in (Tronrud, D.E., Acta Cryst (1999), **A55**, 700-703) Fourier coefficients can be calculated based upon the particulars of the residual function. These coefficients can be fed into Rfactor and the full set of diagonal blocks calculated. This is usually much quicker than calculating the diagonal blocks directly.

A detailed description of the Fourier coefficients which you are required to calculate can be found in the "Creating Your Own Module" chapter.

An example of script for using the NORMAL_MATRIX command is

```
$tntbin/rfactor << $eof
INCLUDE $tntdata/formfactor.dat
INCLUDE your.tnt
INCLUDE init.cor
FILE COEFS  fo-fc.pak FORMAT PACKED
NORMAL_MATRIX rfactor.dat SOURCE COEFS MODULE MY_MODULE
$eof
if ($status >< 0) then exit 1
```

```
PUNCH <File name> {MAP | DSN6 | HKL | PACKED | ATOMx} -

                [TYPE {FO-FC | 2FO-FC | 3FO-2FC | IO-IC | -
                       (FO-FC)**2 | PHASE_GRD | REALSPACE_GRD | -
                       RFACTOR_GRD | ROTATION_GRD | PHASE_CRV | -
                       REALSPACE_CRV | RFACTOR_CRV | -
                       ROTATION_CRV | SWAP_PHASE}] -

                [BLUR <Value>] [SCALE <Value>] -
                [{GRID 3(<value>) | OVERSAMPLE <value>}] -
                [{LAYOUT 6(<value>) | ASYMMETRIC}]
```

The PUNCH command in Rfactor is used to produce files containing either maps or difference coefficients. The modifiers (Scale, Blur, etc.) are described in the "Common Keywords" section of Chapter 3 (page 31).

The type of difference coefficients (either directly written to disk or used to calculate a map) is specified with the TYPE modifier. Most of the coefficient types are self-explanatory. The PHASE_GRD, REALSPACE_GRD, RFACTOR_GRD, and ROTATION_GRD coefficient types define the maps used to calculate the gradients for these modules. The curvatures for these modules can be calculated using the coefficient type with "_CRV" appended to their name. The SWAP_PHASE coefficient type allows the creation of coefficients whose amplitude comes from the FO file while the phase comes from the FC file.

The default file is a 2Fo-Fc map oversampled by 2 and containing a molecular volume layout.

A PUNCH statement must be preceded by a CELL statement, a RESOLUTION statement, the symmetry operators, an FO statement, some coordinates, and the form factors. The structure factors introduced on a FC statement will be used in place of those calculated from the coordinates if a FC statement is given.

If the SIGMA option has been set the coefficients will be sigma weighted. If a SIM statement has been given they will be Sim weighted.

Examples which produce an Fo-Fc and a 2Fo-Fc map are show below. Additional examples are given on page 70.

```
PUNCH   fo-fc.dsn6    DSN6   TYPE FO-FC
PUNCH   2fo-fc.dsn6   DSN6
```

`REPORT {RFACTOR | SPACEGROUP}`

The REPORT statement causes the program to write interesting information to the log file. There are two classes of information which can be reported.

- RFACTOR

  This statement causes the R-value between the two data sets to be printed in the log file. This R-value, along with other interesting facts, are summarized for 10 equal-volume shells in reciprocal space. If the FOM option is specified then additional statistics are printed which are weighted by the Figure of Merit found in the Fo file of structure factors. If the INTENSITY option is set the statistics will be collected over intensity shells. The data will be sigma weighted when the R-value is calculated if the SIGMA option is set.

  An example where RFACTOR is used to produce an R-value is

```
$tntbin/rfactor <<$eof
INCLUDE $refroot/tnt/doc/examples/phrm/tln_phrm.tnt
INCLUDE $refroot/tnt/doc/examples/phrm/phrm.cor
INCLUDE $tntdata/formfactor.dat
REPORT RFACTOR
$eof
if ($status >< 0) then exit 1
```

- SPACEGROUP

  This statement causes Rfactor to write interesting things about the current space group to the log file. The equivalent positions will be listed. Then the point group symmetry and the lattice class will be printed. Finally the centric zones and their phase constraints are printed.

  The only input required for the REPORT SPACEGROUP command is the symmetry operators.

  ```
  $tntbin/rfactor <<$eof
  INCLUDE $tntdata/symmetry/p6122.dat
  REPORT SPACEGROUP
  $eof
  if ($status >< 0) then exit 1
  ```

Program Operation

Creating a Fo-Fo Map

The calculation of a Fo-Fo map requires three pieces of information — the two sets of amplitudes and a set of phases. Since Rfactor works with only two files at a time the phases must be placed in the file containing the standard amplitudes (those being subtracted). If the standard was solved using MIR phasing it is quite convenient to store the MIR phases in the HKL file with the measured amplitudes. When this is the case it is quite simple to calculate a Fo-Fo map. One simply has Rfactor calculate a "Fo-Fc" map where "Fo" is the derivative HKL file and "Fc" is the file containing the standard amplitudes and MIR phases.

If you wish to calculate a Fo-Fo map with phases calculated from a model you must perform the additional step of appending the calculated phases to the standard amplitudes. A file containing these mixed coefficients is produced with the SWAP_PHASE coefficient type. Once this chimeric file is created the Fo-Fo map is calculate as described above.

When the PUNCH is issued with the TYPE SWAP_PHASE modifier the resulting coefficients consist of the amplitudes from the file defined on the FO statement while the phases are from the file defined on the FC file.

K_{sol} is set to zero in either case. This must be done because the disordered solvent contributes to both sets of reflections. K_{sol} is only used when comparing measured amplitudes to calculated ones.

Writing a Fo-Fo Map Phased with MIR Phases

```
$tntbin/rfactor <<$eof
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 2.3
INCLUDE $tntdata/symmetry/p6122.dat

INCLUDE native_model.cor        ! Coordinates included
INCLUDE $tntdata/formfactor.dat ! to provide molecular
                                ! volume.

FO modified_fobs.hkl           ! Amplitudes of new protein
FC native_fobs.hkl             ! Amplitudes and MIR phases

SET KSOL 0.0
PUNCH fo-fc.dsn6 DSN6 TYPE FO-FC OVERSAMPLE 4
$eof
if ($status >< 0) then exit 1
```

Writing a Fo-Fo Map Phased with Calculated Phases

```
$tntbin/rfactor <<$eof
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 2.3
INCLUDE $tntdata/symmetry/p6122.dat

FO native_fobs.hkl              ! Amplitudes of standard
INCLUDE native_model.cor        ! Source of phases
INCLUDE $tntdata/formfactor.dat

PUNCH fobs_phicalc.hkl HKL TYPE SWAP_PHASE

FO modified_fobs.hkl            ! Amplitude of new protein
FC fobs_phicalc.hkl             ! Amplitudes and phases
SET KSOL 0.0
PUNCH fo-fo.dsn6 DSN6 TYPE FO-FC OVERSAMPLE 4
$eof
if ($status >< 0) then exit 1
#
rm fobs_phicalc.hkl
#
```

Chapter 7

Other Modules for TNT

This chapter describes three modules which are implemented in the TNT refinement package, but are not used as often as the GEOMETRY and RFACTOR modules. These modules are:

PHASE:	A module to restrain a model to observed phases
REALSPACE:	A real space refinement module
ROTATION:	A rotation function refinement module

The PHASE and REALSPACE modules require that phase information be present in the observed HKL file. This type of information is usually only needed in the early stages of refinement. Both of these modules appear to be useful when the placement of a known structure in a new crystal is being attempted. Test have been run using rigid body refinement of such structures which show that each of these modules is more powerful than the RFACTOR module.

It is not clear what relative weights should be given to each module when simultaneous refinement of several modules is performed. The user will have to experiment in this area.

Experimental Phase Refinement Module

This module is to be used when one has a set of experimentally determined phases and it is has been decided that the model should be restrained to this information. The function minimized is

$$f(\mathbf{x}) = \sum_{hkl}^{\text{acentric}} \frac{1}{(\cos^{-1} m(\mathbf{s}))^2} (\phi_o(\mathbf{s}) - \phi_c(\mathbf{s}, \mathbf{x}))^2$$

Where $\mathbf{s} = (\,h \quad k \quad l\,)$ and $m(\mathbf{s})$ is the figure of merit of the phase.

It has been found that the gradient of the residual for this term can be calculated using an algorithm similar to Agarwal's method for calculating the gradient of the RFACTOR term. The same convolution is performed but the coefficients used to calculate the map ($C(\mathbf{s})$) are described below.

$$Wt(\mathbf{s}) = \frac{1}{(\cos^{-1} m(\mathbf{s}))^2}$$

$$\Delta\phi(\mathbf{s}) = \phi_o(\mathbf{s}) - \phi_c(\mathbf{s})$$

$$C(\mathbf{s}) = \frac{Wt(\mathbf{s})}{|F_c(\mathbf{s})|} \Delta\phi(\mathbf{s}) \exp i(\phi_c(\mathbf{s}) + \pi/2)$$

As you can see, it is not at all clear from the coefficients what the meaning of the map is. However, because the gradient of the phase residual is calculated from this map using Agarwal's convolution, we know that this map should behave like a normal difference map. There will be positive or negative density centered on an atom if its occupancy or temperature factor is in error and there will be an asymmetric, plus-minus, peak if the position of the atom is incorrect. Because the sources of noise in this kind of map are quite different from those in the usual difference maps, it could very well be profitable to directly examine a map of this form to aid in the interpretation of a model in the early stages of structure determination.

Because of the similarity between the calculations required for this module and those required for the RFACTOR module there are no independent programs for this module. The program Rfactor will calculate all the required values.

Problems

This module does not run as smoothly as the more traditional GEOMETRY and RFACTOR modules. When it is used more short loops are required than one would normally expect. Either there is some error in its implementation or the (completely ignored) complications due to the tight interrelationship between the amplitude and phase of each reflection is the cause. The module does improve the agreement of the calculated and observed phases in spite of its problems.

Example Script

Following is an example script used in the long loop to calculate all the required information for the PHASE and RFACTOR modules. The particular example is from the refinement of native Thermolysin, which has very good MIR phases to 2.3Å resolution. The control file `tln.tnt` contains the cell constants, resolution limits, and the name of the observed structure factor file.

Long Loop Example

```
$tntbin/rfactor << $eof
INCLUDE $tntdata/formfactor.dat
INCLUDE tln.tnt
INCLUDE init.cor
CURVATURE phase.dat   MODULE PHASE
CURVATURE rfactor.dat
$eof
if ($status >< 0) then exit 1
```

Short Loop Example

```
$tntbin/rfactor << $eof
INCLUDE $tntdata/formfactor.dat
INCLUDE tln.tnt
INCLUDE scales.dat
INCLUDE shifted.cor
FUNCTION phase.dat   MODULE PHASE
FUNCTION rfactor.dat
$eof
if ($status >< 0) then exit 1
```

Real Space Refinement Module

Real space refinement has had a long history of being successfully applied to the solution of protein structures. It is useful because it is the simplest way to incorporate both the magnitude and phase of the observed diffraction pattern. Because of the widespread use of reciprocal space refinement methods the real space refinement method has been overlooked in recent years.

The case where real space refinement is indicated is when phases have been determined, by some means, and a partial model has been constructed. To produce a good phase combined map, the partial model should be adjusted to fit the data as closely as possible. Refinement of a partial model in reciprocal space is difficult because it is not clear how much of each observed amplitude is supposed to be due to the missing portion of the model. The problem is solved in real space because the Fourier synthesis which produced the "observed" map brings the relevant density into the neighborhood of the atoms in the partial model.

The function minimized with this module is

$$f(\mathbf{x}) = \sum_{\mathbf{r}}^{\text{space}} (\rho_o(\mathbf{r}) - \rho_c(\mathbf{r}, \mathbf{x}))^2$$

where the calculated electron density map is series terminated to match the resolution limits of the diffraction data. Because the calculated density far from any atom in the model is zero, all terms which depend on density points far from the model are constant and do not affect the gradient of the function. Therefore we only need to sum over the points that are close to the partial model. If one works out the equation for the gradient of this function one will find that it can be calculated using the Agarwal method of the RFACTOR module. The only difference is in the coefficients of the difference map. For this case the coefficients are those of the map which results from the inversion of the Fourier coefficients $F_o \exp i\phi_o - F_c \exp i\phi_c$.

The program Rfactor performs all the work specific to this module. Run the program exactly as you would when running reciprocal space refinement but add a MODULE REALSPACE modifier to the command card.

Problems

While real space refinement has a glorious history, it does not have a very sound mathematical basis. The problem has to do with error analysis. Least squares optimization has built into it the assumption that every observation is independant and its uncertainty follows a Normal distribution. When each density point in the map is considered an "observation" they certainly are not independant – neigboring points tend to have very similar densities.

The real errors of the observations follow a strange distribution. The errors assoicated with the amplitudes are usually much smaller than those associated with the phases. This asymmetry cannot be expressed in real space.

Due to these, or other unknown problems, a cycle of refinement with the real space module usually requires more short loops than expected.

Example Script

Following is given an example script for calculating the function value, gradient, and curvature of this module. It presumes that the control file `tln.tnt` contains the cell constants, space group symmetry, and the name of the file containing the observed structure factors and phases. Since the phases are not reliable to the same resolution of the amplitudes the example script truncates the resolution of the data to be considered. With a lower resolution limit a SET statement must be given to ensure correct scaling.

Long Loop Example

```
$tntbin/rfactor << $eof
INCLUDE $tntdata/formfactor.dat
INCLUDE tln.tnt
RESOLUTION 20 3.0
SET KSOL 0.8
INCLUDE init.cor
CURVATURE realspace.dat   MODULE REALSPACE
$eof
if ($status >< 0) then exit 1
```

Short Loop Example

```
$tntbin/rfactor << $eof
INCLUDE $tntdata/formfactor.dat
INCLUDE tln.tnt
RESOLUTION 20 3.0
INCLUDE scales.dat
INCLUDE shifted.cor
FUNCTION realspace.dat   MODULE REALSPACE
$eof
if ($status >< 0) then exit 1
```

Rotation Function Refinement Module

In Molecular Replacement the orientation of a fragment of known structure is determined by searching for a peak in the “Rotation Function”. Search methods are very useful for avoiding the problems of local minima but are not particularly efficient for getting the exact result. With a solution “close enough” to the peak one can perform least-squares refinement to home in on its center (Yates, T.O., Rini, J.M., Acta Cryst (1990) **A46**, 352–359). This module allows the user to refine the parameters of a model against a rotation function.

Refining with this module is more complicated than the other modules. The data required for the calculation of the function value, gradient, and curvature are different. In addition the space group of the calculation changes from one part of the calculation to another. This module is only implemented under the VMS operating system.

The function minimized with this module is

$$f(\mathbf{x}) = \sum_{\mathbf{s}} \frac{1}{\sigma_I^2(\mathbf{s})}(I_o(\mathbf{s}) - I_c'(\mathbf{s},\mathbf{x}))^2$$

Minimizing this function means that the parameters of the model will be varied to achieve a match between the Patterson coefficients calculated from the model and the observed Patterson coefficients. This function is relatively insensitive to the position of the molecule in the unit cell so the model will not shift in space, it will only rotate. To achieve the magnitude of shifts which are expected for this type of refinement one will usually refine the model as a small number of rigid groups.

There is a complication in the manner in which the space group symmetry is included. Because we do not know the location of the fragment in the unit cell we cannot calculate the Patterson coefficients for the whole unit cell. We must calculate the Patterson coefficients for the model orientated as it would be in each asymmetric unit, with its center at the origin, and sum them. This calculation will eliminate any cross vectors between asymmetric units which is good because we know they are wrong. We then must calculate the gradient of the function for each asymmetric unit in space group P1 and combine the gradients.

The actual procedure is simpler than this explanation sounds because the observed Patterson contains the Patterson group symmetry so the gradient from each asymmetric unit will be equal. We only have to calculate one of them.

An analysis of the function which must be minimized by this module shows that the gradient can be calculated using a variation of the

Agarwal method. The program Rfactor can be used to perform the calculation, only it must be use an alternate set of difference coefficients. These coefficients are

$$C(\mathbf{s}) = 2\frac{(I_o(\mathbf{s}) - I'_c(\mathbf{s}))}{\sigma_I^2(\mathbf{s})}\,|F_c(\mathbf{s})|\,\exp i\phi_c(\mathbf{s})$$

The calculation of the predicted scattering from your model is done in two steps. First the Fourier transform of the model in P1 is calculated. Then the Patterson coefficients ($I'_c(\mathbf{s})$) are calculated in a fashion which excludes contributions from the "cross vectors". These coefficients are calculated with the program Averager_I. The Patterson coefficients are used to calculate the function value and are involved in the gradient calculation.

The function value is calculated by comparing the observed Patterson coefficients with those calculated from the current model. This must be done in the Patterson space group.

The curvature is calculated by comparing the Fobs's with the Fcalc's of the model. Because the model, itself, is in space group P1 the Fobs's must be expanded by its symmetry to P1 so that the reflections can be matched. You must have a copy of your Fobs which has been expanded to space group P1. The atomic parameters are also required for curvature calculation. This module is usually used to perform rigid body refinement. Since one should not use a minimization method which requires the curvatures in rigid body refinement it is not likely that you will need to calculate curvatures.

The gradient is especially complicated to calculate because the coefficients required (the C($\mathbf{s}$) above) depend upon the Fobs's, the Fcalc's, and the Patterson coefficients ($I'_c(\mathbf{s})$). The gradient coefficients must be calculated separately and ran back through Rfactor using the AGARWAL command.

In a short loop all that need be done is to recalculate the Patterson coefficients of the new model, and run RFACTOR again.

Command File Example

Following is given an example command file for calculating the required information for this module. The file names used in this example are those given in the `TNT_EXAMPLES:` area for the Phosphoramidon inhibitor of thermolysin. Since the program Averager_I is VMS specific the example is a VMS command file.

Usually one performs only rigid body refinement with the rotation function module. Therefore this example uses only the low resolution data and does not calculate curvatures.

A complication of this module is that the space group of the calculations changes from place to place. One must be very careful to use the crystal's space group, P1, and the Patterson space group at the proper times. To make the space group clear the example does not use a TNT control file.

```
$ !
$ !  SF's for the inhibitor-enzyme complex in P1.
$ !
$ RUN TNT_UTIL:FOURIER
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 5

INCLUDE TNT_DATA:FORMFACTOR.DAT
INCLUDE INIT.COR
PUNCH TESTFC.HKL HKL
$ !
$ ! Average the Patterson coefficients from our model over
$ !  the Patterson symmetry.  This program will not handle
$ !  the center of symmetry operator in the Patterson
$ !  symmetry.  It must be left out.  This ommission does
$ !  not affect the result of the calculation.
$ !
$ ASSIGN /USER TESTFC.HKL       FOR010
$ ASSIGN /USER TESTFC_I_SYM.HKL FOR011
$ RUN TNT_FFT:AVERAGER_I
```

```
94.1,94.1,131.4,90.,90.,120.,
  x,  y,  z; x-y,  x,  z; -y, x-y,  z;
 -x, -y,  z; y-x, -x,  z;  y, y-x,  z;
x-y, -y, -z;  -y, -x, -z; -x, y-x, -z;
y-x,  y, -z;   y,  x, -z;  x, x-y, -z.
5.0, 20.0
$ !
$ RUN REFROOT:[TNT.RFACTOR]RFACTOR
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 5.0

INCLUDE TNT_SYMMETRY:P6MM.DAT
FO TNT_EXAMPLES:TLN_PHRM.HKL
FC TESTFC_I_SYM.HKL

SET KSOL 0.8
INCLUDE INIT.COR
INCLUDE TNT_DATA:FORMFACTOR.DAT
FUNCTION FUNCTION.DAT MODULE ROTATION
PUNCH COEFS.PAK PACKED TYPE ROTATION_GRD
$ !
$ DELETE        TESTFC_I_SYM.HKL;
$ !
$ ! Expand the fourier coefficients in COEFS.PAK from
$ ! P6MM to space group P1.  One can either use the center
$ ! of symmetry or not.  It does not matter.
$ !
$ ASSIGN/USER   COEFS.PAK       FOR010
$ ASSIGN/USER   COEFS_P1.PAK    FOR011
$ RUN TNT_FFT:EXPANDER
94.1,94.1,131.4,90.,90.,120.,
  x,  y,  z; x-y,  x,  z; -y, x-y,  z;
 -x, -y,  z; y-x, -x,  z;  y, y-x,  z;
x-y, -y, -z;  -y, -x, -z; -x, y-x, -z;
y-x,  y, -z;   y,  x, -z;  x, x-y, -z.
20.0,5.0,
X,Y,Z.
```

```
$ !
$ DELETE COEFS.PAK;
$ !
$ ! Here we have to multiply COEFS_P1.PAK point for
$ !  point with TESTFC.HKL.
$ !
$ RUN TNT_UTIL:FOURIER
FILE FC     TESTFC.HKL   FORMAT HKL
FILE ROTFUN COEFS_P1.PAK FORMAT PACKED
PUNCH ROTATION.PAK PACKED MULTIPLY FC ROTFUN
$ !
$ DELETE COEFS_P1.PAK;, TESTFC.HKL;
$ !
$ ! Calculate the gradient of the rotation function with
$ !  respect to our model.  The model and therefore the
$ !  calculations are in P1.
$ !
$ RUN REFROOT:[TNT.RFACTOR]RFACTOR
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 5.0

INCLUDE TNT_DATA:FORMFACTOR.DAT
INCLUDE INIT.COR
FILE COEFS ROTATION.PAK FORMAT PACKED
AGARWAL GRADIENT.DAT  MODULE ROTATION  SOURCE COEFS
$ !
$ DELETE ROTATION.PAK;
$ !
$ COPY FUNCTION.DAT,GRADIENT.DAT ROTATION.DAT
$ DELETE FUNCTION.DAT;, GRADIENT.DAT;
$ !
$ ! *****  End of modules and into control program ****
$ !
$ RUN REFROOT:[TNT.SHIFT]SHIFT
INCLUDE ROTATION.DAT
INCLUDE OLDDIR.DAT
INCLUDE INIT.COR
```

```
COMBINE   XYZ *
CONSTANT   B
CONSTANT  OCC
$ !
$ ! More clean up.
$ !
$ DELETE        OLDDIR.DAT;, ROTATION.DAT;
$ !
```

This is the command file for the short loops of the same cycle.

```
$ !
$ !  SF's for the inhibitor-enzyme complex in P1.
$ !
$ RUN TNT_UTIL:FOURIER
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 5

INCLUDE TNT_DATA:FORMFACTOR.DAT
INCLUDE SHIFTED.COR
PUNCH TESTFC.HKL HKL
$ !
$ ! Average the Patterson coefficients from our model over
$ !  the Patterson symmetry.  This program will not handle
$ !  the center of symmetry operator in the Patterson
$ !  symmetry.  It must be left out.  This ommission does
$ !  not affect the result of the calculation.
$ !
$ ASSIGN /USER TESTFC.HKL       FOR010
$ ASSIGN /USER TESTFC_I_SYM.HKL FOR011
$ RUN TNT_FFT:AVERAGER_I
94.1,94.1,131.4,90.,90.,120.,
  x,  y,  z; x-y,  x,  z; -y, x-y,  z;
 -x, -y,  z; y-x, -x,  z;  y, y-x,  z;
x-y, -y, -z;  -y, -x, -z; -x, y-x, -z;
y-x,  y, -z;   y,  x, -z;  x, x-y, -z.
```

```
5.0, 20.0
$ !
$ DELETE TESTFC.HKL;
$ !
$ RUN REFROOT:[TNT.RFACTOR]RFACTOR
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 5.0
INCLUDE TNT_SYMMETRY:P6MM.DAT
FO TNT_EXAMPLES:TLN_PHRM.HKL

FC TESTFC_I_SYM.HKL
INCLUDE SCALES.DAT
FUNCTION ROTATION.DAT  MODULE ROTATION
$ !
$ DELETE TESTFC_I_SYM.HKL;
$ !
$ ! Get rid of the old SHIFTED.COR.
$ !
$ DELETE        SHIFTED.COR;
$ !
$ ! Now move the atoms as directed.
$ !
$ ASSIGN/USER   SHIFT.OUT       SYS$OUTPUT
$ RUN REFROOT:[TNT.SHIFT]SHIFT
INCLUDE STPFIL.DAT
INCLUDE ROTATION.DAT
INCLUDE INIT.COR
INCLUDE NEWDIR.DAT
$ !
$ TYPE  SHIFT.OUT
$ !
$ DELETE ROTATION.DAT;
$ PURGE  STPFIL.DAT
```

Chapter 8

Creating Your Own Module

To allow TNT to restrain a macromolecular model to a new type of observation you will need to write a program. You will have to define the precise function you plan to minimize. Your program will have to be able to calculate the value of this function and its first derivative with respect to each parameter of the model, given any coordinate set. To allow yourself the benefits of the preconditioned conjugate gradient method the program should also be able to calculate the second derivatives. TNT only uses atomic block diagonal elements of the normal matrix. The list of required second derivatives consists of a symmetric 5x5 block for each atom.

Passing Information to TNT

To be compatible with the TNT programs your program will need to read and write standard TNT files. The format of the coordinate and sequence files have been defined in the chapter "Shared I/O Properties" chapter and Appendix A of this manual and "TNT Sequence File" chapter of the TNT Users' Guide. The only unfamiliar keywords will be those used to pass the function value and derivatives to Shift.

Six keywords are provided to allow information to be passed from the various modules to the control program. They are called FUNC-

VAL, DRVx, GRADx, CRVC, CRVAC, and CRVBC. All of these statements are free-format like most everything else in TNT. Remember the definition of <Atom designator> is

<Atom name> <Residue name> <Chain name>

```
CRVC <Module name> <Term curvature> <Atom designator>
```

$$< \text{Term curvature} >:== \\ < \partial^2 f/\partial X^2 >< \partial^2 f/\partial Y^2 >< \partial^2 f/\partial Z^2 >< \partial^2 f/\partial B^2 >< \partial^2 f/\partial Occ^2 >$$

This statement is used to pass the curvature of each parameter for the atom mentioned to the control program. Curvatures are always be specified in the TNT orthogonal coordinate system. If you do not calculate these values, do not write these statements and the minimizer will revert to a noncurvature minimization method.

```
CRVAC <Module name> <Term A curvature> <Atom designator>
```

$$< \text{Term A curvature} >:== \\ < \partial^2 f/\partial X\partial Y >< \partial^2 f/\partial Y\partial Z >< \partial^2 f/\partial Z\partial B > - \\ < \partial^2 f/\partial B\partial Occ >< \partial^2 f/\partial X\partial Z >$$

This statement is used to pass some off-diagonal elements of each atom's curvature matrix for the atom mentioned to the control program. Curvatures are always be specified in the TNT orthogonal coordinate system. If you do not calculate these values, do not write these statements and the minimizer will revert to a pure diagonal approximation to the Normal matrix.

```
CRVBC <Module name> <Term B curvature> <Atom designator>
```

$$< \text{Term B curvature} >:== \\ < \partial^2 f/\partial Y \partial B >< \partial^2 f/\partial Z \partial Occ >< \partial^2 f/\partial X \partial B > - \\ < \partial^2 f/\partial Y \partial Occ >< \partial^2 f/\partial X \partial Occ >$$

This statement is used to pass some off-diagonal elements of each atom's curvature matrix for the atom mentioned to the control program. Curvatures are always be specified in the TNT orthogonal coordinate system. If you do not calculate these values, do not write these statements and the minimizer will revert to a pure diagonal approximation to the Normal matrix.

```
DRV  <Module name> <Term gradient> <Atom designator>
DRVC <Module name> <Term gradient> <Atom designator>
DRVG <Module name> <Term gradient> <Atom designator>
```

$$< \text{Term gradient} >:== \\ < \partial f/\partial X >< \partial f/\partial Y >< \partial f/\partial Z >< \partial f/\partial B >< \partial f/\partial Occ >$$

These statements are used by the various modules to pass the term's derivatives for each atom in the structure to the control program. The derivatives for every atom must be supplied by every module being used in TNT.

```
FUNCVAL <Module name> <Value>
```

One FUNCVAL statement is generated by each module and passed to the control program. This statement contains the name of the module in question and the present value of the function which that module deals with.

Using TNT to Calculate Derivatives

While the calculation of the gradient and curvature usually is not too difficult for many restraint types when diffraction gets involved these calculations can become difficult to code in an efficient manner. TNT contains tools which can be used to speed these calculations and lessen the amount of work you have to do to implement your ideas.

The Agarwal method of calculating gradients of the usual least-squares restraint is generalizable to any restraints which are a function of amplitudes and phases. In this method the gradient of a residual function is calculated by generating Fourier coefficients which depend upon the form of the residual, and processing the corresponding map in a general fashion. For the calculation of the gradient the coefficients are given to the AGARWAL command in Rfactor. The diagonal of the Normal matrix is calculated in a similar fashion by passing different coefficients to the NORMAL_MATRIX command.

The Fourier coefficients required to calculate the gradient of a residual of the form

$$f = \sum_{\mathbf{s}} G(\mathbf{s}) \tag{8.1}$$

is

$$C(\mathbf{s}) = -\frac{1}{2}\left(\frac{\partial G(\mathbf{s})}{\partial|F_c(\mathbf{s})|} + i\frac{1}{|F_c(\mathbf{s})|}\frac{\partial G(\mathbf{s})}{\partial\phi_c(\mathbf{s})}\right)\exp i\phi_c(\mathbf{s}). \tag{8.2}$$

The coefficients required to calculate the curvature of a residual is much more complicated. The principle component can be calculated by passing

$$C(\mathbf{s}) = \frac{1}{2}\left(\frac{\partial^2 G(\mathbf{s})}{\partial|F_c(\mathbf{s})|^2} - i\frac{1}{|F_c(\mathbf{s})|^2}\frac{\partial G(\mathbf{s})}{\partial\phi_c(\mathbf{s})}\right) \tag{8.3}$$

to the NORMAL_MATRIX command.

Chapter 9

Refinement Control Program (Shift)

The program Shift performs all the calculations required to oversee the minimization of the function. It can determine the optimum shift to apply to the parameters of the model, while imposing various constraints defined by the user. The model, for example, can be broken into one or more groups which move as rigid bodies. Also particular parameters can be held fixed.

Shift always works with the starting coordinate set (`init.cor`) and a current fraction of shift, called the stepsize. When the stepsize is equal to zero (i.e. no shift has been applied) the program works in "long loop" mode. Conversely, when the stepsize is non-zero the program performs the "short loop" calculations.

In either case, the program evaluates all available information and determines its best guess of the next stepsize which should be tried. It then generates a new trial set of coordinates (`shifted.cor`) by applying this guess. The only files required to do this task are the initial coordinates (`init.cor`), the parameter shift vector (`newdir.dat`), the function value for the current stepsize (usually `rfactor.dat` and `geometry.dat`), and the history of the previous guesses (`stpfil.dat`).

In a short loop that is all that is done. In the long loop the direction of shift must first be determined. The calculation of the parameter shift

	Curvature	
`olddir.dat`	Absent	Present
Absent	Steepest Descent	Gradient / Curvature
Present	Conjugate Gradient	Preconditioned Conjugate Gradient

Table 9.1: Choice Table for Minimization Method

vector requires, in the simplest case, the first derivatives of the function being minimized. The inclusion of the second derivatives improves the quality of the shift vector considerably. The shift vector must be modified to ensure that the user's constraints will not be violated.

Shift can use four different methods to calculate the parameter shift vector. These methods are discussed in detail in the chapter "Theory of TNT Refinement" (page 7). The program is never told which method to use. It simply looks at the information which has been given to it and uses the best method it can. The optional data are the old shift vector (`olddir.dat`) and the second derivatives of the function. Table 9.1 shows the rules used to determine the method.

Options

Shift recognizes two options. They can be set with the OPTION statement described in the "Standard TNT Data Statements" section on page 27. As an example, the activation of the VERBOSE option would require the statement

```
OPTION SET VERBOSE
```

- MOSES

  The Moses option is used to modify the behaviour of solvent atoms in rigid-body refinement. Normally any atoms not specifically listed on COMBINE statements are left as free atoms. If you model contains solvent atoms and several COMBINE statements you would have to assign each solvent atom to a particular statement, which can be time-consuming and error prone. When the Moses option is SET, the waters are "split" between the COMBINE statements by assigning each atom to the COMBINE statement which contains its nearest neighbor. This option does not affect only solvent, all atoms are assigned to the closest COMBINE statement resulting in the restriction of all free atoms.

- VERBOSE

  The Verbose option causes the program to write additional information to the log file about the choices it makes. This includes a summary of the function value for each module and the actual parameters of the curve being locally fit to the overall function.

Unique Input Statements

Shift has several unique input statements which are only used in the passing of information from the programs performing the function value and derivative calculations. These statements are discussed in the "Writing Your Own Module" chapter (page 87).

In addition, it understands:

```
MAXSHIFT <Parameter code> <RMS limit>
```

The MAXSHIFT statement allows a limit to be placed on the size of any shift for any atom. While the RANGE statement limits the allowed values of a parameter, MAXSHIFT limits the allowed range of shifts which may be applied. <Parameter code> is either XYZ, B, or OCC (ALL is not allowed). The <RMS limit> times the root-mean-square of all shifts gives the upper limit for the size of any particular shift. If a shift is greater than this limit it is clipped to that limit.

(The r.m.s. shift is determined in a robust fashion; all shifts greater than 4 times the r.m.s. are ignored.)

MAXSHIFT is only useful when using steepest descent or conjugate gradient. When the shifts are not scaled by the curvatures some parameters will be vastly overshifted, especially temperature factors of heavy atoms. The MAXSHIFT statement causes the resulting oscillations to damp much quicker. When using a minimization method which utilizes curvature, such as preconditioned conjugate gradient there is no need to use this statement.

```
RMSTEP   <Parameter code> <RMS shift>
```

The RMSTEP statement gives Shift an initial amount of shift to be applied. The parameter code identifies whether a RMS coordinate (XYZ), B factor (B), or occupancy (OCC) is being defined. The short loops will refine this number to find the best value. Because this number is refined its choice is not critical but values 1 1/2 to 2 times larger than the expected final value seem to work best. Default values exist for the RMSTEP statement so that in a long loop it need not to be given.

RMSTEP is ignored whenever the Gradient/Curvature or Preconditioned Conjugate Gradient methods are used. In both of these cases the curvature is used to determine the initial stepsize.

Parameter Editing Statements

```
COMBINE <Parameter code> N(<Atom code>)
```

This statement creates a subset of the structure that will be treated as a rigid body. The positional shifts of each atom in this class are combined so that the relative positions of the atoms within the group are unaffected by the shifts to be applied. Several ridge bodies can be refined simultaneously by entering several COMBINE statements. Each COMBINE statement generates a single rigid body.

If non-positional parameters are combined, the shift applied to every atom is the average of the shift that would have been applied to each atom. Therefore if all of the atoms in the subset began with the same value then they would maintain values equal to each other, even though the particular value would change.

When positional parameters are combined the group is allowed to rotate and translate as a unit.

The details of this statement are described in the “TNT Control File” chapter of the “TNT Users’ Guide”.

```
CONSTANT <Parameter code> N(<Atom code>)
```

This statement sets the shifts for a specified subset of the structure to zero causing the specified parameters to remain constant.

The details of this statement are described in the “TNT Control File” chapter of the “TNT Users’ Guide”.

Program Operation

This program is quite simple to run. All you have to do is provide the information from each module, and describe any modifications to the shifts you wish to make. The most common type of refinement uses the crystallographic and stereochemistry modules. Usually we do not have high enough resolution to determine the occupancies of the atoms in the molecule. The occupancies will have to be held fixed. The data the program needs in the long loop is:

```
$tntbin/shift << $eof
INCLUDE geometry.dat
INCLUDE rfactor.dat
INCLUDE init.cor
INCLUDE olddir.dat
CONSTANT OCC
$eof
if ($status >< 0) then exit 1
```

Usually the CONSTANT, COMBINE, and other parameter editing statements are placed in the TNT control file. The CONSTANT statement here is placed in the script for illustrative purposes only.

The information from each module is contained in `rfactor.dat` and `geometry.dat`. These files must contain, at least, the function value and gradient of their respective terms. If the files also contain the curvature Shift will be able to use the more powerful methods of function minimization.

The initial coordinates are contained in the file `init.cor`. The shift vector, or direction file, is contained in `olddir.dat`. This file is the shift applied in the last cycle. During the first cycle of a series there will not be an `olddir.dat`, therefore you should ensure that an empty one exists. Its presence will keep Shift from reporting a “file not found” error message.

The occupancies of all atoms are held fixed. Because this is the only CONSTANT statement all the positions and temperature factors will be varied.

The input to SHIFT during a short loop in this same cycle looks like:

```
$tntbin/shift << $eof
INCLUDE stpfil.dat
INCLUDE newdir.dat
INCLUDE init.cor
INCLUDE geometry.dat
INCLUDE rfactor.dat
$eof
if ($status >< 0) then exit 1
```

Note that `stpfil.dat` and `newdir.dat` are now included. `stpfil.dat` contains the history of this cycle while `newdir.dat` contains the current shift vector. `olddir.dat` is no longer needed. It should be deleted at the end of the long loop.

Again `init.cor` is read. Shift will always work relative to the initial coordinates.

The TNT control file is not required in the short loops.

Information from the individual modules is again passed in `geometry.dat` and `rfactor.dat`. However in the short loops these files need only contain the function values.

The CONSTANT statements should not be reentered in the short loops because the knowledge that they had been entered in the long loop has been stored in `newdir.dat`.

Chapter 10

Refinement Package Utilities

There are several utility programs distributed with this refinement package. They are of variable quality and usefulness and not all are described in this document. The programs described are:

Convert	A coordinate format conversion program
Dscreen	A derivative file screening program
Fourier	A Fourier transformation utility
Gather_map	Calculates ncs averaged maps
Overlay	Superimposes molecules or subsets on each other
Solvent	Calculates solvent envelopes and flattened maps

CONVERT

Coordinate Conversion Program

This program is used to convert coordinate files both to and from the standard formats of this refinement package. It can perform any of the conversions in the following table.

From	*To*
ATOM	ATOM
ATOMC	ATOMC
ATOMG	ATOMG
SEQUENCE	SEQUENCE
DIAMOND	DIAMOND
DSN2	DSN2
HENDRICKSON	HENDRICKSON
PDB	PDB
	AMSOM

As shown in the table Convert can produce a sequence file given a DIAMOND, DSN2, HENDRICKSON, or PDB format file. The sequence is inferred by a method that involves the CATEGORY, DANGLING, and CONNECT statements. The sequence file produced will have to be edited to handle unusual circumstances. This program does not change atom names to adapt to different naming conventions. However a file containing CHANGE commands which will correct most atom type problems is supplied. It is named `$tntdata/pdb_fixup.dat`

This program has several additional functions. It can alter the name of a variety of objects with the RENAME command (e.g. atoms, residues, chains). It can change the type of an atom, residue, or chain with the CHANGE command. It can add a particular value to all the temperature factors or occupancies of the model or add a vector to the positional parameters with the ADD command. And it can add a random quantity with specified variance to any of these parameters with the JIGGLE command.

Conversion Limitations

There are a number of fundamental problems with format conversions of coordinate files. These problems arise because concepts used in one format may not be defined in another.

Residue Names

The most important of these difficulties is that in some formats, such as the PDB format, residues are identified with numbers, while TNT's format uses the full alphanumeric character set in residue names. While the conversion from PDB format to ATOMC is quite simple in this respect the reverse operation is not clearly defined. The solution used by Convert is to simply write the original residue names in the PDB file. This choice easily leads to illegal PDB files.

Atom Types

Another common problem with coordinate file format conversion involves atom types. TNT defines atom types with four letter names. PDB format uses the one or two letter standard element abbreviation. This means that a PDB file cannot define the ionic state of an atom, which is important when calculating the X-ray scattering from that atom.

The first two columns in the atom name field are reserved for the elemental type in PDB format. Knowing this convention usually allows Convert to at least identify the element correctly. However, the PDB does not follow its own rules. The standard atom labels chosen for a number of cofactors do not begin with the element symbol for that atom. For instance, the C4$'$ atom in the ribose ring in ATP is labeled AC4*. Convert will misidentify this atom, and almost all the other atoms in an ATP, as having the elemental type "A". To correct the problem you will need to INCLUDE the file `$tntdata/pdb_fixup.dat` between the PDB and PUNCH commands.

Diamond and DSN2 formats use integer codes to identify atom types. Unfortunately the definition of the formats do not specify an atom type code for any atom types other than carbon, nitrogen, oxygen, and sulfur. Beyond those atom types your have to create your own standard. Convert allows you to specify the atom type code for each atom type.

For the file formats Diamond and DSN2 you can specify, on the end of the PUNCH command, a table defining the integer code for each atom type found in your model. When reading back the Diamond file you must include this same table of conversion values on the DIAMOND statement. Convert hides this information in the DSN2 format so you are not required to reenter the table when reading back a DSN2 file.

Multiple Chains

A problem which arises with DSN2 format is that there is no method for including multiple chains in the file. When Convert writes a multiple chain model to DSN2 format it prepends the first letter of the chain name to each residue's name. Therefore residue "ALPHA|1" becomes "A1" in the DSN2 file. Convert hides the information that "A" is denoting chain "ALPHA" in the DSN2 file so that the proper chain name can be restored when converting back to ATOMC format.

The PDB file format allows no more than a single letter to identify a chain. TNT will allow up to four letters but since your model will have to be convertible to PDB format you should stick to single letters.

Chain Breaks and Extra Links

Often a peptide chain in a model, and sometimes in reality, is broken. Residue 123 might directly follow residue 120 in the coordinate file without a peptide bond connecting the two. Usually, in PDB, Diamond, and DSN2 formats, it is implicitly assumed that each residue is linked to the next. When this does not happen Convert will err and link them. When you are first creating a sequence file you should be aware of the breaks in the main chain of your model and edit the sequence file to include this information.

Non-TNT formats do not include ability to define other kinds of links between residues. If you have non-standard crosslinks in your molecule you will have to edit the sequence file for these as well.

DNA Connectivity

TNT can use the statements described in the next section to generate a nearly correct sequence file for DNA. There are some problems which must be manually corrected.

The first problem is that the TNT geometry library requires that the first two bases in DNA be connected via the d5'END link instead of the normal dSUGPHOS link. If this correction in not made TNT will complain about the absence of some atoms which are not supposed to be present. The progress of refinement will not be affected.

The second problem is that TNT cannot distinguish between DNA and RNA. The bases in DNA are connected using the dSUGPHOS link in the sequence file while RNA bases should be connected using the SUGPHOS link. If your model contains RNA you will have to edit the sequence file.

Keywords for Inferring the Sequence

These statements are used to define the assumptions about the connectivity implicit in other file formats. They are required for Convert to construct a TNT sequence file. A file containing the most common assumptions is supplied with the TNT distribution. This file is named `$tntdata/connect.dat`. It should be INCLUDEd prior to the inclusion of any foreign coordinate file, such as a PDB or DSN2 file, if a sequence file is to be PUNCHED. (This is done in the shell command **from_pdb**.)

The basic idea is that collections of residue types can be treated identically. A collection of these residue types is defined on a CATEGORY statement. For example, in terms of connectivity all amino acids are considered the same. Therefore we define a category of

AMINOACID to consist of the twenty amino acid types (some types have synonymous names) with the statement

```
CATEGORY   AMINOACID   VAL,   GLY,   ALA,   SER,   LEU,   -
                       ILE,   PHE,   TRP,   PRO,   CPR,   -
                       MET,   THR,   CYS,   CYH,   TYR,   -
                       ASN,   GLN,   ASP,   GLU,   LYS,   -
                       ARG,   HIS,   MSE
```

We then specify what type of link should be generated when an AMINOACID follows another AMINOACID with a CONNECT statement. This statement looks like

```
CONNECT  AMINOACID  AMINOACID  PEPTIDE
```

If an amino acid follows an amino acid they should be connected with a PEPTIDE linkage.

Finally we define with a DANGLING statement what to do when a residue of a particular category is followed by nothing. A dangling amino acid should be connected to an empty residue with a BREAK linkage. This requirement is translated to

```
DANGLING  AMINOACID  BREAK
```

`CATEGORY <Category name>  N(<Residue type>)`

This statement is used to place a group of residue types into a category. All the residues in a category will be treated the same when the connectivity of a molecule is deduced.

```
CATEGORY  BASE   ADE, GUA, CYT, THY
```

This example defines the residue types ADE, GUA, CYT, and THY to be in the category BASE.

`CONNECT <Category name> <Category name> <Linkage type>`

The CONNECT statement says that whenever a residue of the first category is followed by one of the second, they are to be linked with the given linkage type. This mechanism will not add any secondary linkages, like disulfide bonds, but can build a basic peptide backbone.

CONNECT BASE BASE dSUGARPHOS

This example says that when a BASE follows a BASE they should be joined by a dSUGARPHOS linkage. Of course, you would want to say something different if your molecule contained RNA instead of DNA.

`DANGLING <Category name> <Linkage type>`

The DANGLING statement is used when the residues of a particular require a link even when they are not followed by anything. Both amino acids and nucleic acids have this requirement. When the sequence file is being created and a dangling residue is discovered a dummy residue is manufactured and the residue is linked to the dummy via the linkage type given on the DANGLING statement.

DANGLING BASE d3'END

This example says that the terminal BASE should be connected to a dummy residue with a d3'END linkage.

Unique Keywords for Specifying Data

```
DIAMOND <File name> N(<Atom type code>  <Atom type name>)
```

This statement allows the program to read the atomic coordinates from a Diamond format file. The file name of the Diamond file is the first item after the keyword. It is followed by a list of translation pairs. These pairs describe how the integer atom types of the Diamond file should map onto the atom type names that this package uses to identify atoms. The pairs (1 C), (2 N), (3 O), (4 S), and (5 P) are built into the program. If there are no other atom types in your structure and the above translation pairs are correct then you don't need to worry about atom type codes at all. Just give the file name and nothing else.

A CELL statement must precede a DIAMOND statement. If a sequence file is desired the file `$tntdata/connect.dat` must be INCLUDEd before this statement.

The coordinate system of the atoms in the Diamond file is assumed to be crystallographic Ångstroms (fractional times the cell edges). No other coordinate system of DIAMOND file can be read or written.

Here are some examples:

DIAMOND gewl.rdi
DIAMOND phase5r.rdi 12 MG++

The first example tells the program to read the file `gewl.rdi` and to use only the normal translation table. The second statement tells the program to read the file `phase5r.rdi` and to add another atom type to the translation table. The pair of values says that atom type 12 corresponds to the atom type MG++.

```
DSN2 <File name> N(<Atom type code> <Atom type name>)
```

This statement allows the program to read the atomic coordinates from a Frodo DSN2 format file. The file name of the DSN2 file is the first item after the keyword. It is followed by a list of translation pairs.

These pairs describe how the integer atom types of the DSN2 file should map onto the atom type names that TNT uses. The pairs (1 C), (2 N), (3 O), (4 S), and (5 P) are built into the program. If there are no other atom types in your structure and the above translation pairs are correct then you don't need to worry about atom type codes. Just give the file name and nothing else.

If the DSN2 file was created by TNT the file will contain all the information required to translate the integer atom codes into TNT style atom type names. You do not need to enter any additional information.

If the DSN2 file does not contain cell constant information the DSN2 statement must be preceded by a CELL statement. If a sequence file is desired the file `$tntdata/connect.dat` must be INCLUDEd before this statement.

Here are some examples:

```
DSN2 gewl.dsn2
DSN2 phase5r.dsn2 12 MG++
```

The first example tells the program to read the file `gewl.dsn2` and to use only the normal translation table. The second line tells the program to read the file `phase5r.dsn2` and to add an additional atom type. The entry says that atom type 12 corresponds to the atom type MG++.

`HENDRICKSON <File name>`

This statement causes the program to read the named file into the internal data base. This allows the input of Hendrickson format coordinates. A CELL statement is required in all cases and `$tntdata/connect.dat` should be INCLUDEd if a sequence file is desired. The name of the atom type of each atom is assumed to be the first character of the atom name. If there are atoms in the structure whose type cannot be determined by this method you will either have to use the RENAME command in your script or edit the file.

```
PDB <File name>
```

This statement causes the program to read the named file into the internal data base. This allows the input of Protein Data Bank format coordinates. Cell constant information is read from the file so a CELL statement not is required. The file `$tntdata/connect.dat` must be INCLUDEd before the PDB statement if a sequence file is desired. The elemental type of each atom is determined from the first two columns of the atom name field. While this method usually gives the correct elemental type it cannot give the ionic state of the atom. This information will have to be given either using the RENAME command or manually editing the final coordinate file.

While `$tntdata/connect.dat` should be INCLUDEd prior to the PDB statement it is usually wise to INCLUDE the file `$tntdata/pdb_fixup.dat` afterwards. This file will correct a wide range of possible problems, including the correction of most elemental types.

Convert will automatically change the C-termini from PDB style to TNT style. It will move the extra oxygen atom into a residue of its own and link it with a CTERM linkage. Convert will also read the disulfides off the SSBRIDGE statements and include then in the new sequence file.

Convert will not properly create a sequence file for DNA or RNA, although it will be close. See Page 103 for details.

Command Statements

`ADD {XYZ | B | OCC} {<Value> | <Value> <Value> <Value>}`

This command is used to add a constant to a particular type of parameter for each atom in the structure. When the XYZ parameters are to be affected three values must be given to define the translation vector. When either temperature factors (B's) or occupancies (OCC's) are specified a single number is given. In any case the number read from the statement is added to each parameter of the indicated type in the model. For instance this command may be used to change the average temperature factor of a model.

No B factor or occupancy can be moved outside the values imposed by the most recent RANGE statement.

This operation would be performed with a statement looking like

```
ADD   B   5.0
```

`JIGGLE {XYZ | B | OCC} <RMS value>`

This command adds a random value to the parameter class specified. The amount of distortion is normally distributed with the r.m.s. value specified by the user. The values allowed for B or OCC parameters are be restricted to a particular range by the most recent RANGE statement. By default B factors will lie between 1.0 and 100.0 while OCC parameters will be restricted to values between 0.0 and 1.0.

Some example statements are given here.

```
JIGGLE   XYZ   1.0
JIGGLE   B     5.0
```

The first example mangles the XYZ parameters of the model with an r.m.s. value of 1.0. The second example adds an r.m.s. variation of 5.0 to the B factors.

```
PUNCH <File name> {ATOM | ATOMC | ATOMG | SEQUENCE | CELL |
                   AMSOM | PDB | DIAMOND |
                   DSN2 | HENDRICKSON} -
       N(<Atom type code>  <Atom type name>)
```

This command writes out all atoms presently known to the program to the named file in the format specified.

A CELL statement is required to convert to Diamond, DSN2, Hendrickson, or PDB formats. DSN2 file conversion also requires the symmetry operators of the space group.

If Amsom, PDB, Diamond, DSN2, or Hendrickson format is desired, the sequence file for the structure must have been supplied. If a SEQUENCE file is desired, a sequence must be entered, either by giving a sequence file (the null operation), or a PDB, Diamond, DSN2, or Hendrickson formatted file.

One would want to PUNCH a file of CELL format when converting from a PDB format file. The cell constants are usually stored in the PDB file. One can PUNCH these values in a file which contains only a CELL statement. This file can be INCLUDEd when other TNT programs read the new coordinate file.

Here are some examples.

```
PUNCH   test.pdb    PDB
PUNCH   test.dsn2   DSN2      20 CA++   30 ZN++
PUNCH   test.rdi    DIAMOND   20 CA++   30 ZN++
```

The coordinate system of the Diamond file produced is crystallographic Ångstroms (fractional times cell edges). This coordinate system is the only one that can be produced in a Diamond file with Convert.

Program Operation

The most common operations performed by Convert are to convert to and from PDB format, and to and from Frodo's DSN2 format. An example of each operation is given below.

Converting PDB to ATOMC format

```
$tntbin/convert << $eof
INCLUDE $tntdata/connect.dat
PDB pdb1tmn.ent
INCLUDE $tntdata/pdb_fixup.dat

PUNCH 1tmn.cor ATOMC
PUNCH 1tmn.seq SEQUENCE
$eof
if ($status >< 0) then exit 1
```

This example reads the PDB file named `pdb1tmn.ent` and produces the TNT coordinate file `1tmn.cor` and the sequence file `1tmn.seq`. If you do not need a sequence file simply delete the second PUNCH command. This example will work for any PDB file. Since the shell command **from_pdb** can be used to perform this conversion you do not need to write one of your own.

ATOMC to PDB format

```
$tntbin/convert << $eof
CELL 94.1 94.1 131.4 90 90 120
INCLUDE 1tmn.cor
INCLUDE 1tmn.seq

PUNCH 1tmn.pdb PDB
$eof
if ($status >< 0) then exit 1
```

This example writes the coordinates and sequence information stored in `1tmn.cor` and `1tmn.seq` to the PDB file `1tmn.pdb`. The **to_pdb** shell command performs this function.

Converting DSN2 to ATOMC format

```
$tntbin/convert << $eof
INCLUDE $tntdata/connect.dat
DSN2 tln_phs5r.dsn2  15 P 17 CL- 30 ZN++ 20 CA++

PUNCH tln_phs5r.cor ATOMC
PUNCH tln_phs5r.seq SEQUENCE
$eof
if ($status >< 0) then exit 1
```

This example reads the DSN2 file named `tln_phs5r.dsn2` and produces the TNT coordinate file `tln_phs5r.cor` and the sequence file `tln_phs5r.seq`. (The atom type translation table is unneeded if the DSN2 file was produced by Convert.) If you do not need a sequence file simply delete the second PUNCH command. This example will work for any DSN2 file. Since the shell command **from_dsn2** can be used to perform this conversion you do not need to write one of your own unless you have special atom type codes.

Converting ATOMC to DSN2 format

```
$tntbin/convert << $eof
CELL 94.1 94.1 131.4 90 90 120
INCLUDE 1tmn.cor
INCLUDE 1tmn.seq

PUNCH 1tmn.dns2 DSN2  15 P 17 CL- 30 ZN++ 20 CA++
$eof
if ($status >< 0) then exit 1
```

This example writes the coordinates and sequence information stored in `1tmn.cor` and `1tmn.seq` to the DSN2 file `1tmn.dsn2`. The shell command **to_dsn2** performs this function.

DSCREEN

Derivative File Screening Program

This program is used to find out which atoms are least satisfied with their current parameters, with respect to a particular module. It reads the derivative file produced by the module of interest, finds the largest derivatives for the three classes of parameters; XYZ, B, and OCC, and writes a nicely formatted list of them. It is intended that this list be used to aid in the interpretation of the map when work is being done to manually rebuild a model.

The refinement of a model has converged when none of the parameters are changing. However, a parameter can become fixed for two reasons; either all the modules agree that this is a good value, or each module prefers a different value and the parameter is balanced in between. The second case is a sign that manual intervention is required to place the atoms in that region in a new local minimum. This program is a tool for finding which regions of the model are in trouble. Where the various modules are at odds, the derivatives from each module will be larger than normal and point in opposite directions. This utility will read the derivative file and point out the problem areas in the model.

The program requires that the coordinates of the model, and the derivative file be entered before the SCREEN command. A CELL and a GRID statement may be required if the fraction or grid coordinate system is used for either the coordinates or the derivatives.

Command Statements

```
SCREEN <Number>
```

This command causes the program to produce a list of the atoms with the largest derivatives for each class of parameters. An example showing the use of the SCREEN command is shown below.

```
$tntbin/rfactor << $eof
INCLUDE $tntdata/formfactor.dat
INCLUDE your.tnt

INCLUDE init.cor
OPTION SET REFINEMENT
GRADIENT rfactor.dat
$eof
if ($status >< 0) then exit 1
#
$tntbin/dscreen << $eof
INCLUDE init.cor
INCLUDE rfactor.dat
SCREEN 45
$eof
if ($status >< 0) then exit 1
#
rm rfactor.dat
```

FOURIER

Diffraction Data Conversion Program

This program performs various operations on density maps and Fourier coefficients. It can convert one form of data into the other and calculate either from atomic coordinates. It can also combine such data from two sources to produce a new map or coefficient file. The data may be either added, subtracted, multiplied, or divided.

The program operates on three different forms of data: Coordinate sets, density maps, and Fourier coefficients. Its purpose is to perform interconversions between these types of data. The conversion between maps and coefficients requires a Fourier transform. The conversion of coordinates to a density map is a straightforward calculation of the density of each atom. The conversion of a map to coordinates returns an atom located at each peak in the map. This is the familiar peak pick operation.

When a map is calculated from coefficients those coefficients are weighted by their figure of merits to produce the “best” map. If no FOM’s are given no weighting is performed.

Most of the work of this program is performed with the PUNCH command.

The Space Groups

When calculating coefficients from a map or set of coordinates all space groups are implemented. This includes space groups with cubic symmetry, centers of symmetry, and mirrors. When calculating a map from coefficients all space groups work except for those which contain mirrors but do not contain centers of symmetry (such as Pm). This means that maps can be calculated in all space groups in which a protein can crystallize.

In all cases rhombohedral space groups must be indexed in the hexagonal setting.

Options

FOURIER recognizes one option. This option can be set with the OPTION statement described in the "Standard TNT Data Statements" section. As an example, the activation of the VERBOSE option would require the statement

OPTION SET VERBOSE

- VERBOSE

  The Verbose option causes the program to write additional information to the log file about the choices it makes.

Unique Input Statements

```
FILE <File Id> <File name> [FORMAT (HKL | MAP | PACKED)] -
                           [SCALE <Value>] [BLUR <Value>]
```

This statement describes an external binary data file to the program. This file is subsequently referred to by the arbitrary "File Id". If the file's format is not HKL, which is the default, the type of file may be given via the FORMAT option. When the data are read from the file a scale factor and blur may be applied. Because of the extreme difficulty of applying a blur to an electron density map, when the file is a map file, the BLUR option is presumed to indicate that the map has a already been blurred by the indicated amount.

Command Statements

```
PUNCH <File name> {MAP | DSN6 | HKL | PACKED | ATOMx} -

        [{SOURCE <Id> | CORRECT <Id> | -
          ADD      <Id1> <Id2> | SUBTRACT <Id1> <Id2> | -
          MULTIPLY <Id1> <Id2> | DIVIDE  <Id1> <Id2> |
          PATTERSON <Id>}] -

        [HIGHEST <Value> ] -

        [BLUR <Value>] [SCALE <Value>] -
        [{GRID 3<Value> | OVERSAMPLE <Value>}] -
        [{LAYOUT 6<Value> | ASYMMETRIC}]
```

One may PUNCH data in the formats:

Short Name	Full Name
MAP	Ten Eyck Map Format
DSN6	Frodo DSN6 Map Format
HKL	Formatted Coefficients
PACKED	Unformatted Coefficients
ATOMx	Identifies peaks in map

The rest of the PUNCH command line describes the origin of the information and the details of the file to be produced. The various modifiers (SCALE, BLUR, etc.) are described in the "Standard TNT Data Statements" section (page 31).

The default coefficient file is an asymmetric unit of coefficients calculated from the atomic positions.

The default map is a molecular volume of model electron density written in Ten Eyck map format, sampled at a rate proper for representing such a density function.

What kind of map?

There are five types of sources for the information in a file. If no source is explicitly mentioned the program uses the atomic coordinates for the source. This is how model electron density and model structure factors are produced.

The second type of source is introduced via the SOURCE keyword. This keyword is followed by the file identifier of a map or coefficient file which is processed to produce the new file. If the data type of the input file differs from that of the output file the proper data type conversion is performed.

The CORRECT keyword is the third type. It behaves just like the SOURCE keyword except that the coefficients read, or calculated from, the file are moved to the correct asymmetric unit and sorted. Usually this operation is only required when a data set of observed reflections are being converted to a TNT format. In such a case you would read the illegal HKL file with the CORRECT keyword and write out a new HKL file. One could read a map file with the CORRECT keyword but that would be wasteful, the code which converts the map to structure factors will automatically produce data in the correct asymmetric unit and sorted in the proper order.

CORRECT also restricts the asymmetric unit of the data to include only unique reflections. All other types of sources in TNT will produce a small number of redundant reflections.

The fourth type of input is controlled with the ADD, SUBTRACT, MULTIPLY, and DIVIDE keywords. These keywords are followed by two file identifiers. The data in these two files are read and each transformed to the output data type. Then the contents are combined using the specified operator and written to the file.

One must use these keywords with care. It is quite likely that the operation you request will change either the space group, the resolution limit of the data, or both.

The last type of "source" is controlled by the PATTERSON keyword. This keyword indicates that the data to be written will be either

Patterson coefficients or a Patterson map, depending upon the output file's format.

How much map to write?

While the default map layout is a molecular volume this can be overridden. With the keyword ASYMMETRIC one can request an asymmetric unit of density. A more specific request can be made using the LAYOUT keyword.

These keywords are meaningless when writing Fourier coefficients.

How large will the map file be?

When writing a map the program will choose a sampling rate, the grid, for the map. The sampling rate is chosen based on the highest resolution Fourier component of the map. While this is simple when the map is to be calculated from Fourier coefficients it is another matter when calculating density from atomic coordinates. In this case the map will have to be sampled very finely because the atoms contain such high resolution data. One can reduce the size of the map by blurring out the high resolution terms you are not interested in. This is done with the BLUR option. When you use the BLUR option you must remember to indicate to the program reading the map that it has been blurred, because the map must be sharpened when read.

Fourier will never write a map with a non-zero blur unless you specifically request it. It is usually wise, when writing calculated electron density maps to blur them with a value of 20Å^2.

Picking Peaks from a Map

One can enter either a set of Fourier coefficients or a map and request that Fourier list the peaks in that map. Fourier will locate each peak and attempt to deduce the parameters of an atom which would generate a peak of that height. Fourier will always produce an atom with an occupancy of 1.0. The temperature factor will be chosen to replicate the height of the peak in the map. The position of the peak will be the nearest grid point. If possible Fourier will produce atoms of type "O". If the temperature factor of an oxygen atom must be below 1.0 to generate a peak of the needed height heaver atom types will be tried.

The default region of space for difference map peaks is a molecular volume. Of the many possible symmetry images of a peak, one is chosen which is closest to an atom in the current coordinate set. If Fourier has been given no coordinates the peaks will be placed in the real-space asymmetric unit. The peaks in a arbitrary region of space can be selected using the GRID and LAYOUT modifiers of the PUNCH command.

When PUNCHing peaks the additional modifier 'HIGHEST' is available. In the default case all peaks in the map are listed to the output file. One can use the HIGHEST modifier to ask that only the tallest peaks are produced. The number following the word HIGHEST is the number of peaks to list.

The conversion from peak height to temperature factor is very sensitive to errors in the scaling of the observed and calculated data, as well as to the presence of the low resolution components of the map. A high resolution refined difference map which includes all the low resolution data will provide good estimates of the temperature factors of the atoms identified. As the map becomes poorer the estimates will develop greater errors. The tendency will be to overestimate the magnitude of the B factor.

Most people do not realize it but the scale of a Fo-Fc map is different that that of a 2Fo-Fc map. A Fo-Fc map must always be multiplied by a factor of two to be placed on an absolute scale of electrons/$Å^3$. If you have a refined model and choose one water molecule to study you may find, for example, that the density at the center of the atom in a model map is 2.5 electrons/$Å^3$. If you remove the atom from the coordinate file and calculate a Fo-Fc map you will find the peak height is only 1.25. (This is exemplified in the derivation of the 2Fo-Fc map coefficients. One starts with a Fc map. Next a difference map (Fo-Fc) is added but because the Fo-Fc map is on a different scale it must be multiplied by two before the addition. The result is Fc + 2(Fo-Fc) or 2Fo-Fc.) In almost all uses of the peak pick function the input coefficients will be difference coefficients, and they will have to be multiplied by two. This operation is performed using the SCALE modifier on the FILE statement which introduces the map or coefficient file. The example below (way below!) may make this issue more clear.

Example of the Calculation of Structure Factors in $P6_3$

This example will convert a list of coordinates to a set of Fourier coefficients.

```
$tntbin/fourier << $eof
CELL 111.9 111.9 98.2 90 90 120
RESOLUTION 20 2.5
INCLUDE $tntdata/symmetry/p63.dat
INCLUDE $tntdata/formfactor.dat
INCLUDE init.cor
PUNCH testfc.hkl HKL
$eof
if ($status >< 0) then exit 1
```

To calculate a model electron density map instead the PUNCH command in the the example should be replaced with

PUNCH test.map MAP ASYMMETRIC BLUR 20

This command will write to the file `test.map` an asymmetric unit of electron density with a blur of 20Å^2. The sampling rate will be chosen to be the coarsest allowed for this blur.

Converting an HKL File to a PACKED File

This example converts a data set from HKL format to PACKED format. Because there is no data type conversion the cell constants and symmetry operators are not required.

```
$tntbin/fourier << $eof
FILE MIRPHASE bcl.hkl FORMAT HKL
PUNCH mir.pak PACKED SOURCE MIRPHASE
$eof
if ($status >< 0) then exit 1
```

Converting a MAP File to an HKL File

This example converts a data set from MAP format to HKL format. An asymmetric unit of map must be supplied in `density.map`.

```
$tntbin/fourier << $eof
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 1.7
INCLUDE $tntdata/symmetry/p6122.dat
FILE MAP density.map FORMAT MAP
PUNCH coefficient.hkl HKL SOURCE MAP
$eof
if ($status >< 0) then exit 1
```

Converting a PACKED File to a DSN6 Map File

This example will convert a set of Fourier coefficients to DSN6 map format. The coefficients are in the file `2fo-fc.pak`. The coordinate file `current.cor` is read simply to allow the layout of the DSN6 map to be determined by the model's molecular volume.

```
$tntbin/fourier << $eof
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 1.7
INCLUDE current.cor
INCLUDE $tntdata/symmetry/p6122.dat
FILE COEFS 2fo-fc.pak FORMAT PACKED
PUNCH 2fo-fc.dsn6 DSN6 SOURCE COEFS OVERSAMPLE 4
$eof
if ($status >< 0) then exit 1
```

Locating the Peaks in a Map

This example locates the peaks in a difference map. Because map files are so large it is inefficient to read and write them. It is always better to use the PACKED Fourier coefficient file format to transfer the information instead. Here the difference coefficients are to be found in the file `fo-fc.pak` and the current model is located in `current.cor` The 30 tallest peaks are written to `peaks.cor`.

```
$tntbin/fourier << $eof
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 20 1.7
INCLUDE current.cor
INCLUDE $tntdata/symmetry/p6122.dat
INCLUDE $tntdata/formfactor.dat
FILE FO-FC fo-fc.pak FORMAT PACKED SCALE 2.0
PUNCH peaks.cor ATOMC SOURCE FO-FC HIGHEST 30
$eof
if ($status >< 0) then exit 1
```

Adding Two MAP Files

The new file is created by adding `map_1.map` and `map_2.map`.

```
$tntbin/fourier << $eof
FILE FIRST_MAP  map_1.map FORMAT MAP
FILE SECOND_MAP map_2.map FORMAT MAP
PUNCH new.map MAP ADD FIRST_MAP SECOND_MAP
$eof
if ($status >< 0) then exit 1
```

Note the the adding of two maps does not require that you enter the space groups of either map. However you should be aware that the resulting map will have the space group that is the greatest common subgroup of the space groups of the two arguments. If you ask for the Fourier coefficients of the result of the summation TNT will use the proper space group in the calculation.

Correcting the Asymmetric Unit of Coefficients

```
$tntbin/fourier << $eof
CELL 94.1 94.1 131.4 90 90 120
RESOLUTION 1.5
INCLUDE $tntdata/symmetry/p6122.dat
FILE FOBS bad_asymm.hkl FORMAT HKL
PUNCH good_asymm.hkl HKL CORRECT FOBS
$eof
if ($status >< 0) then exit 1
```

This example (which is basically the code for the shell command **correct**) moves the data from the file `bad_asymm.hkl` to the correct asymmetric unit, removes all redundant reflections, notes all inconsistent duplicate reflections, sorts the remaining ones into the desired order, and writes the resulting data set to `good_asymm.hkl`.

```
REPORT {FILE <Id> | SPACEGROUP}
```

- FILE

  The REPORT FILE command instructs the program to list interesting things about a map or coefficient file on disk. Among other things, it will list the r.m.s. value for a map file or the r.m.s. value for the map which would result from a file of coefficients. The program will also list the Wilson B factor for a coefficient file. The calculation of the Wilson B requires the scattering factors for the atom types present in the model.

  ```
  $tntbin/fourier << $eof
  CELL 94.1 94.1 131.4 90 90 120
  INCLUDE $tntdata/symmetry/p6122.dat
  RESOLUTION 1.5
  INCLUDE $tntdata/formfactor.dat
  ```

```
FILE FOBS pgn.hkl  FORMAT HKL
REPORT FILE FOBS
$eof
if ($status >< 0) then exit 1
```

For Fourier coefficient files this report is accessed using the **report_hkl** shell script.

- SPACEGROUP

  The REPORT SPACEGROUP command causes Fourier to write its analysis of the current space group. It lists the equivalent positions, the lattice class, the point group symmetry, and the centric zones.

```
$tntbin/fourier << $eof
INCLUDE $tntdata/symmetry/p6122.dat
REPORT SPACEGROUP
$eof
if ($status >< 0) then exit 1
```

GATHER_MAP

NCS Map Averaging Program

This program is used to generate maps which have been averaged over the noncrystallographic symmetry. This operation is required when your model is being constrained to ncs and you are starting model building.

Like all constrained ncs in TNT, Gather_map is a little clunky. The script for running it must be edited for each project, and therefore not generic script is supplied. The prototype script can be found in `tnt/doc/gather_map` and will be described later. You will be required to calculate a separate map for each CLUSTER in your molecule. While on the graphics system you will need to remember to switch from one map to another at the proper time.

You cannot use ncs map averaging programs from other packages along with TNT refinement. TNT's idea of noncrystallographic symmetry is more general than all other programs in the field. Other programs define the transformation as a rotation and a translation. TNT's ncs tranformation is generalized to include a rotation, translation and a B factor shift. When an ncs averaged map is calculated, the individual images must be rotated, translated, and blurred or sharpen the appropriate amount. It does not make sense to average two maps, one of a molecule with an average B of $20Å^2$ and another with an average B of $40Å^2$. The second map must be sharpened by $20Å^2$ prior to being added to the first.

Options

Gather_map recognizes one option. This option can be set with the OPTION statement described in the "Shared I/O Properties" chapter (page 27). As an example, the activation of the VERBOSE option would require the line

OPTION SET VERBOSE

- VERBOSE

  The Verbose option causes the program to write additional information to the log file about the choices it makes.

Unique Input Statements

```
FILE <File Id> <File name> {FORMAT (HKL | MAP | PACKED)} -
                           [SCALE <Value>] [BLUR <Value>]
```

This statement describes an external binary data file to the program. This file is subsequently referred to by the arbitrary <File Id>. If the file's format is not HKL, which is the default, the type of file may be given via the FORMAT option. When the data are read from the file a scale factor and blur may be applied. Because of the extreme difficulty of applying a blur to an electron density map, when the file is a map file, the BLUR option is presumed to indicate that the map has a already been blurred by the indicated amount.

Command Statements

```
PUNCH <File name> {MAP | DSN6 | HKL | PACKED} -

        CLUSTER <Cluster name> -

        [BLUR <Value>] [SCALE <Value>] -
        [{GRID 3<Value> | OVERSAMPLE <Value>}] -
        [{LAYOUT 6<Value> | ASYMMETRIC}]
```

(The details of the PUNCH command are described in the "Shared I/O Properties" chapter – page 31.)

The part of the PUNCH command unique to Gather_map is the CLUSTER modifier. This modifier introduces the name of the CLUSTER statement which will be used to locate the ncs transformations. The default layout covers the atoms of the cluster's prototype.

You will note that the file ID of the source coefficients (or map) is not given. Here we come to a klunky part. To calculate the averaged map, Gather_map must calculate all the unaveraged maps at the same time. Unfortunately a limitation in the internal design of TNT is that a particular file ID can only be read for one map calculation at a time. Even though there is only one input file there must be a seperate file ID for each copy of the molecule in the asymmetric unit. This requirement can result in a large number of FILE statements.

Since the PUNCH command does not allow you to specify the file ID of your choice you must use the file ID's it assumes. These ID are not very creatively named. They are MAP01, MAP02, etc. Your script must contain one FILE statement for each copy of the molecule in your asymmetric unit.

Program Operation

The only command in the program Gather_map is PUNCH. Most of the details were covered in the previous section. Here we will concentrate on an example. Let's calculate ncs averaged 2Fo-Fc maps for the monoclinic form of Hen Egg White Lysozyme as described in the "Examples" chapter of the TNT Users' Guide. This crystal has two molecules in the asymmetric unit, and each molecule is composed of two domains. We will calculate an averaged map for each domain. The starting point is a "gathered" coordinate file.

Since the coordinate file contains all the atoms in the asymmetric unit are required to directly calculate the 2Fo-Fc Fourier coefficients we must first "scatter" the model, and then calculate normal 2Fo-Fc coefficients. Gather_map will take these coefficients, the "gathered" coordinates, the ncs transformations (here assumed to be in the file `gathered.ncs`, and the usual crystallographic information (cell constants, etc.) and produce the map. The script looks like this. (Usually one would enter the TNT control file but I have included the crystallographic informaiton directly to make its presence more obvious.

```
$tntbin/gather_map << $eof
CELL 28 62.9 60.5 90 90.8 90
INCLUDE $tntsymmetry/p21.dat

RESOLUTION 2.2

INCLUDE pdb1lym.seq
CLUSTER N RESIDUE 1 - 91 CHAINS A B
CLUSTER C RESIDUE 92 - OXY CHAINS 1 B

INCLUDE gathered.cor
INCLUDE gathered.ncs

FILE MAP01 2fo-fc.pak FORMAT PACKED
FILE MAP02 2fo-fc.pak FORMAT PACKED

PUNCH 2fo-fc_n.dsn6 DSN6 CLUSTER N OVERSAMPLE 4
PUNCH 2fo-fc_c.dsn6 DSN6 CLUSTER C OVERSAMPLE 4
$eof
if ($status >< 0) exit 1
```

In this example the layout of the two maps are identical. Each covers all the atoms present in `gathered.cor`. Each averaged map is only vaild over the atoms in the chain with the name of its cluster, i.e. the map in `2fo-fc_n.dsn6` is only valid for atoms in the N chain prototype.

OVERLAY

A Molecular Superposition Program

This program is used to superimpose two chemically identical sections within a molecule or in two different molecules. To use this program one first specifies a target with the TARGET statement. The target is a portion of the structure which will not be moved while other sections of the structure are placed on top of it. The target may be either a single residue or a residue range. Only the atoms within each residue whose names are on the SELECT statement will be used to determine the match. A SELECT statement must always be given and all the atoms on the SELECT statement must be in each of the residues mentioned on both the TARGET and OVERLAY statements.

The command OVERLAY is used to specify a residue or residue range for which a rotation matrix and translation vector will be calculated (Kabsch, W., Acta Cryst. (1978). **A34**, 827-828). The OVERLAY command does not apply this transformation to anything, it simply is remembered. The APPLY command is used to ask the program to apply the transformation to a specified portion of the structure. The portion may either be a residue, or a list of wild cards. The PUNCH command causes the new coordinate set to be printed to the specified file in the desired format. The PUNCH command can also be used to print the transformation itself to a file.

This program will accept and apply transformations defined with the TRANSFORMATION statement. The TRANSFORMATION statement will accept Euler angles, spherical polar angles, or a raw matrix. When multiple OVERLAY and TRANSFORMATION statements are encountered the most recent is used by the APPLY and PUNCH commands.

The program can also RENAME a chain or chain type, or COPY a collection of atoms to a new chain. These commands allow a great deal of flexibility in the operation of this program. With the RENAME command one can read two structures which originally had the same chain name and then superimpose them.

Unique Input Statements

```
SELECT N(<Atom name>)
```

This statement allows a number of atom names to be read. When the subject is being matched to the target, only atoms whose names are on a SELECT statement will be used. Each residue considered must contain all the atoms mentioned on the SELECT statement. This statement is required for the OVERLAY command.

Each SELECT statement adds its atoms to the complete list. Once atoms have been selected they cannot be unselected.

```
SELECT   CA
SELECT   CA C N O
```

The first example shows the statement that would be given if one wished to overlay two protein molecules but only to consider the alpha carbons. The second line would be used for the same problem if it were decided to use all the main chain atoms.

```
TARGET <Chain name> | <Residue 1> - <Residue 2>
TARGET <Chain name> | <Residue name>
```

This statement specifies the target over which the portion of the structure selected on the OVERLAY statement will be superimposed. The specified portion of the structure must be either a single residue or a residue range. It must match the OVERLAY structure residue for residue. No wild cards are allowed in any of the fields.

Command Statements

```
APPLY <Chain name>|<Residue name>:<Atom name>
```

This command instructs the program to apply the most recent transformation to the specified portion of the structure. Wild cards may be placed in any of the fields. If nothing follows the APPLY keyword the transformation is applied to all of the structure.

```
COPY <Chain name>|<Residue name>:<Atom name> <Chain name>|
```

This command copies the atoms specified to the second chain mentioned on the input line. The atoms are not simply moved from one chain to the other, they are replicated. Wild cards may be placed in the <Residue name> or <Atom name> slots. The source chain must exist and the destination chain must not. The sequence of the residues in the new chain is preserved. However, any interchain linkages specified on the CHAIN statement are not copied to the new chain because there is insufficient information to build the new links.

```
OVERLAY <Chain name>|<Residue 1> - <Residue 2>
OVERLAY <Chain name>|<Residue name>
```

This command causes the program to calculate the transformation which, when applied to the specified section of the structure will cause it to be superimposed on the target residue range specified on the last TARGET statement. If the number of residues given do not match the number on the TARGET statement or there is not a one-to-one match of atoms in the two residue ranges (after considering all SELECT statements) the program will exit with an error message. At least one SELECT statement is must have been encountered before the OVERLAY command.

```
PUNCH <File name> {ATOM | ATOMC | ATOMG | TRANSFORMATION}
```

The PUNCH command causes the program to write the coordinates of the entire structure into the named file in the specified format.

If the TRANSFORMATION format is specified, a TRANSFORMATION statement will be written to the named file. This statement will contain the current transformation in TNT's cartesian Ångstrom coordinate system.

```
RENAME {CHAIN <Chain name> | CHAIN_TYPE <Chain type>} -
                                              TO <New name>
```

This statement causes the name of the specified chain or chain type to be changed to the value specified. This sort of operation is useful when one wants to compare two structures but both structures are defined using the "chain with no name". The first structure can be read, its chain name changed, and then the second structure can be read.

```
RENAME   CHAIN        null   TO 157I
RENAME   CHAIN_TYPE   null   TO 157I
```

This example shows how a structure, defined with the default chain can be renamed so that both the chain type and chain name is 157I. (In cases where the chain name and type are unnamed, as is usually done when there is only one chain, you can reference the unnamed object with the name "null".

The RENAME command can rename several other kinds of things as well. For the full details see page 32.

Program Operation

Superimposing ncs Related Chains

```
$tntbin/overlay << $eof
INCLUDE azurin.cor
INCLUDE azurin.seq
SELECT CA C N O
TARGET  1|1 - 129
OVERLAY 2|1 - 129
APPLY   2|
PUNCH output.cor ATOMC
$eof
if ($status >< 0) then exit 1
```

This example show how one chain of a structure can be overlayed onto another. The structure used is that of Azurin solved by Dr. E. N. Baker. The coordinate and sequence files are included and the main chain atoms are selected. The target is defined as all 129 amino acids of chain 1. The transformation which superimposes the 129 residues of chain 2 onto chain 1 is then determined. Next this transformation is applied to all of chain 2 and the resulting coordinates are written with a PUNCH command.

Superimposing Residues within a Model

```
$tntbin/overlay << $eof
INCLUDE core5r.cor
SELECT C1A NA C4A CHB C1B C2B C3B CAB C4B NB
SELECT CHC C1C NC C4C CHD C1D C2D C3D CAD C4D ND CHA
TARGET 360
OVERLAY 362
APPLY   362
OVERLAY 364
APPLY   364
OVERLAY 366
APPLY   366
```

```
OVERLAY 368
APPLY   368
OVERLAY 370
APPLY   370
OVERLAY 372
APPLY   372
PUNCH output.cor ATOMC
$eof
if ($status >< 0) then exit 1
```

This example is more complicated. Here Overlay reads in the coordinates of the bacteriochlorophyll core of the bacteriochlorophyll-a containing protein, selects for consideration only the atoms in the conjugated ring system, superimposes each ring, in turn, onto ring 360, and writes the resulting coordinates to the file `output.cor`.

Symmetry Expansion

Next is shown how to use Overlay to expand a structure by crystallographic symmetry. The example is the Bacteriochlorophyll-a containing protein which exists as a trimer but the asymmetric unit is a monomer. Sometimes it is useful to expand the coordinates out to the full trimer.

```
$tntbin/overlay << $eof
CELL 111.9 111.9 98.3 90 90 120
TRANSFORMATION three-fold OPERATOR -Y+1, X-Y, Z
INCLUDE bcl_phase7r.cor
RENAME CHAIN NULL  TO A

COPY    A|      B|
APPLY   B|

COPY    B|      C|
APPLY   C|

PUNCH test.cor ATOMC
```

```
$eof
if ($status >< 0) then exit 1
```

The symmetry operator and the coordinates are read. Just to clean up the appearance of the final coordinate file the original nameless chain is renamed to A. Then chain A is copied to B and B is transformed. Then B is copied to C and it too is transformed by the same operator. C, having been transformed twice, is now right where we want it.

Applying Arbitrary Transformations

This last example shows how to apply an arbitrary transformation to an entire coordinate set. The coordinates are read in and the transformation is specified on a TRANSFORMATION statement. The transformation is then applied with the APPLY command and the result is PUNCHed out.

```
$tntbin/overlay << $eof
INCLUDE phase5r.cor
TRANSFORMATION XXXX SYSTEM TNT -
     MATRIX -0.5 -0.866 0 0.866 -0.5 0 0 1 -
     TRANSLATION 96.9082 -55.95 0
APPLY
PUNCH output.cor ATOMC
$eof
if ($status >< 0) then exit 1
```

SOLVENT

Solvent Region Flattening Program

This program is used to perform the calculations required for solvent flattening of poorly phased electron density maps. It can be used to calculate a mask of the solvent region and it can apply this mask to the an electron density map to produce a map where the protein region is unaffected but the solvent region is absolutely flat.

The method used to calculate the solvent mask is that of Wang (Wang, B.C., "Resolution of Phase Ambiguity in Macromolecular Crystallography", Methods Enzymol. (1985). **115**, 90-112) as modified, to improve performance, by Leslie (Leslie, A.G.W., "A Reciprocal-space Method for Calculating a Molecular Envelope using the Algorithm of B. C. Wang", Acta Cryst. (1987). **A43**, 134-136). First the map is read and the density values less than zero are set to zero. This map is Fourier inverted and the coefficients are multiplied by a function whose own Fourier transform is a cone with a particular radius. This operation is the equivalent of a weighted local average of the map.

A new map is calculated from the modified coefficients. A level of electron density is chosen so that a certain percentage of the density points in the map are below it. These points define the solvent region of the unit cell. The percentage is the solvent fraction of the crystal and must be supplied by the user. The points in the solvent region are marked with a value of zero, which produces the envelope. This envelope can be used to "mask" out the solvent region of the original map or can be written to the disk for later use.

The radius of the cone used in the local averaging step depends on the size of the molecule and the quality of the data. The default value is 9.25Å, but this value may be overridden by the user. At the start you will want to examine the solvent envelope to judge its quality. It should be connected with few to no holes. You can change the envelope by varying the radius. A larger radius will produce a smoother envelope while a smaller radius will produce a more elaborate one.

Options

Solvent recognizes one option. This option can be set with the OPTION statement described in the "Shared I/O Properties" chapter (page 27). As an example, the activation of the VERBOSE option would require the line

OPTION SET VERBOSE

- VERBOSE

  The Verbose option causes the program to write additional information to the log file about the choices it makes.

Unique Input Statements

```
FILE <File Id> <File name> {FORMAT (HKL | MAP | PACKED)} -
                           [SCALE <Value>] [BLUR <Value>]
```

This statement describes an external binary data file to the program. This file is subsequently referred to by the arbitrary <File Id>. If the file's format is not HKL, which is the default, the type of file may be given via the FORMAT option. When the data are read from the file a scale factor and blur may be applied. Because of the extreme difficulty of applying a blur to an electron density map, when the file is a map file, the BLUR option is presumed to indicate that the map has a already been blurred by the indicated amount.

```
SET [RADIUS <Value>] [SOLVENT <Value>] [LEVEL <Value>]
```

The SET statement is used to define the values of the parameters used in calculating the solvent envelope and the flattened map. The definitions of these parameters are:

RADIUS	The radius of the averaging cone (in Å).
SOLVENT	The percent of the crystal which is solvent.
LEVEL	The density value the flattened region is set to.

RADIUS is to be given in Ångstroms and LEVEL should be defined in whatever units the other density values in the map have, usually electron/Å^3.

Command Statements

```
PUNCH <File name> {MAP | DSN6 | HKL | PACKED} -

        (ENVELOPE <Id> | FLATTEN <Id>) -

        [BLUR <Value>] [SCALE <Value>] -
        [{GRID 3<Value> | OVERSAMPLE <Value>}] -
        [{LAYOUT 6<Value> | ASYMMETRIC}]
```

(The details of the PUNCH command are described in the "Shared I/O Properties" chapter – page 31.)

There are two types of maps produced by Solvent. If the source of information (defined with the file identifier ID) is introduced by the modifier ENVELOPE the map produced by the PUNCH command will be a molecular envelope. If the modifier is FLATTEN the program will produce a flattened map and write that to disk.

There are two maps required to produce a flattened map. One needs the map whose solvent region is to be obliterated and one needs a solvent envelope to define that solvent region. The default operation of Solvent is to calculate the solvent mask from the map which is to be flattened. If you wish to utilize a different solvent mask you may introduce this mask to Solvent by defining the file identifier ENVELOPE to correspond to the desired file. If the file identifier ENVELOPE has been defined that file will be assumed to contain the proper solvent mask.

Program Operation

The only command in the program Solvent is PUNCH. With it you can create an envelope marking the solvent region or a solvent flattened map. The examples given below assume that the TNT control file contains the cell constants, and the space group's symmetry operators. For clarity the resolution limits and SET statements are shown explicitly.

Example of Calculating a Solvent Envelope

```
$tntbin/solvent << $eof
INCLUDE your.tnt
RESOLUTION 3.0
FILE DATAFILE density.map FORMAT MAP
SET SOLVENT 60.0
PUNCH envelope.map MAP ENVELOPE DATAFILE GRID 144 144 128
$eof
if ($status >< 0) then exit 1
```

Example of Calculating a Flattened Map

```
$tntbin/solvent << $eof
INCLUDE your.tnt
RESOLUTION 3.0
FILE MAP init.map FORMAT MAP
SET SOLVENT 60 LEVEL 0.0
PUNCH flat.map MAP FLATTEN MAP
$eof
if ($status >< 0) then exit 1
```

Example of Calculating a Flattened Map using Another Mask

```
$tntbin/solvent << $eof
INCLUDE your.tnt
RESOLUTION 3.0
FILE MAP init.map FORMAT MAP
FILE ENVELOPE envelope.map FORMAT MAP
SET SOLVENT 60 LEVEL 0.0
PUNCH flat.map MAP FLATTEN MAP
$eof
if ($status >< 0) then exit 1
```

Appendix A

File Formats

ATOMx Format Coordinate Files

In this format, there is one line per atom. The line starts with the word ATOM for fractional crystallographic coordinates, ATOMC for Cartesian coordinates in Ångstroms, or ATOMG coordinates in crystallographic grid points, followed by the atom type, the atomic coordinates, the individual temperature factor, the occupancy, the atom name (i.e. CA, O, NE1, etc.), the residue name, and chain name (optional) in that order. All alphanumeric fields (atom type, atom name, residue name, and chain name) are left justified. The programs in this package will read a coordinate file with any spacing between elements on a line. To allow the user to write programs to read and write coordinate files using Fortran FORMAT statements the following format is adhered to – (A6,A4,3(1X,F9.4),1X,F6.2,1X,F6.4,1X,A6,2(1X,A4)). The program Convert and can be used to punch a coordinate file in this format if given an arbitrarily spaced ATOM, ATOMC, or ATOMG file.

They also can change coordinate files from one file format to another. Both of these functions are carried out by the PUNCH command.

The coordinate system used for the Cartesian system uses the three axes a^*, $c \times a^*$, c (**This is not the PDB's convention**). The program Geometry will display the orthogonalization and deorthogonalization matrixes for any crystal when given a CELL statement followed by the command REPORT STRUCTURE.

REMARK statements may be placed anywhere in the coordinate file. They usually will be echoed to the output device when read and are not passed to any other file.

Here is an example of the format.

```
ATOMC N      34.6140   40.2998   -0.9950  25.56 1.0000 N      1
ATOMC C      33.6483   41.1420   -0.2133  24.46 1.0000 CA     1
ATOMC C      33.9352   42.6028    0.0604  24.32 1.0000 C      1
ATOMC O      35.0805   43.0812    0.2754  25.61 1.0000 O      1
ATOMC C      33.5180   40.4243    1.2352  22.76 1.0000 CB     1
```

DSN2 (Frodo) Format Coordinate Files

DSN2 coordinate files can only be written and read by the TNT program Convert. DSN2 files are binary files which are read using direct record access. The format was designed to allow the rapid location of the atoms of particular residues. It presumes that the entire file will not be read into memory. The assumptions which underlie these choices are no longer valid and one expects that no new graphics programs will be written to read this format.

The file format itself is not documented well. The implementation in TNT was created by reverse-engineering the file as produced by Alwyn Jones' program TOM. It appears that Turbo-Frodo uses a different definition. Turbo-Frodo's DSN2 is not supported by TNT.

If you are interested in the details of TNT's implementation you can write to me and I will send them to you.

HKL Format Structure Factor Files

This type of file has one reflection per line, in the following format. The line should begin with the keyword "HKL ", followed by h, k, l, amplitude, sigma, phase (in degrees ranging from 0 to 359.9), and figure of merit in the format (A4, 3I4, 3F8.1, F8.4). If the phase for this reflection is unknown the figure of merit should be zero and the phase should be 1000. The flag of 1000 for a phase is redundant and often not used but one must insure that the figure of merit is zero when no phase information is present.

```
REM  This is an example HKL file.
HKL    0   0   6  1076.0     0.0   180.0  1.0000
HKL    0   0  18   443.0     0.0     0.0  1.0000
HKL    0   0  24   374.0     0.0     0.0  1.0000
HKL    0   0  30  1083.0     0.0     0.0  1.0000
HKL    0   0  36   254.0     0.0   180.0  1.0000
```

Any number of remarks can be placed at the beginning of the file. Each remark statement must begin with "REM " instead of "HKL ".

The reflections must be sorted with l varying the most rapidly and h varying the least rapidly. All programs that read this type of file require that the format spacing be exact. The TNT shell script **correct** will change the hkl values of an HKL file to move the reflections to the asymmetric unit TNT requires as well as sort the reflections.

PACKED Format Structure Factor Files

PACKED coefficient format is a dense unformatted file structure designed to minimize disk space usage. At the beginning of the file is a block of comments. Like a map file each comment is 80 bytes in length and all but the last comment begins with an asterisk ("*"). A minimum of one comment is required. Following the comment block is a sequence of records each composed of two parts. The first part is an integer array 100 elements long and the second part is a complex array 100 elements long. The record that is composed of these two array contains up to (surprize!) 100 reflections. The integer array contains the indices for the coefficient stored in the corresponding element of the complex array. The indices are packed using the formula:

$$\text{Packed Index} = ((h + 256)512 + k + 256)512 + l + 256$$

This formula causes the restriction that no h, k, or l can fall outside the range of -255 to 256. I have not had the good fortune to have any problems with this limitation but virus people might.

Any element in the array of packed indices with a value of 0 should be ignored. An element of -1 signals the end of the list of reflections and no more reads should be performed on the file.

Ten Eyck MAP Format Files

Ten Eyck map files are unformatted files. They are used to store various density maps. The file begins with a variable number of comment records 80 characters long. All comment records begin with an asterisk ("*") except for the last comment record.

After the comments follow one or more map sections, each composed of two records. The first record of a map section contains eight integer numbers. The first is the grid value for this section. The next two give the starting and ending grid value in grid points along the fast direction, then the next two numbers give the range in the slow direction. The last three numbers give the sampling rate along each axis (fast first, then slow, and finally section).

The second record contains

$$\text{(FAST<end> - FAST<start> + 1)*} \\ \text{(SLOW<end> - SLOW<start> + 1)}$$

real numbers which is the map section. The End of File is detected when the header record for a section contains only -1's. There will be no records following this header record.

There is a standard convention for which axes are "fast", "slow", and "section". When the map file contains Z sections, X is fast and Y is slow. For Y sections, Z is fast and X is slow. It rarely arises but when X sections are desired Y is fast and Z is slow. In general the preference is for Z sections. Ideally any program which reads Ten Eyck map files should accept any type of sectioning, but this ideal is rarely achieved.

Alwyn Jones' DSN6 Map Format Files

The DSN6 map format is designed to allow the rapid location of blocks (called bricks in this context) of density for display on a graphics system. The file in binary with a fairly simple structure.

It is composed of a series of records which are all 512 bytes long. The first is a header which contains all the information required to interpret the rest of the file. The subsequent bricks contain blocks of electron density. Each density sample is one byte in size. The parameters for mapping real numbers to bytes is stored in the header. The order of the samples within a brick is "x fast, y medium, and z slow". This means that for the first value of y and z all values of x are written. Then y is incremented and all the x values for the new (y,z) pair are written.

The order of bricks within the file is the same. For the first values of y and z all the x bricks are written out in order.

The data in the header is composed of 256 integer*2 values, most of which are never used. Those which are are

(1)	x start
(2)	y start
(3)	z start
(4)	x extent
(5)	y extent
(6)	z extent
(7)	x sampling rate
(8)	y sampling rate
(9)	z sampling rate
(10)	Header(18) * A Cell Edge
(11)	Header(18) * B Cell Edge
(12)	Header(18) * C Cell Edge
(13)	Header(18) * $\cos\alpha$
(14)	Header(18) * $\cos\beta$
(15)	Header(18) * $\cos\gamma$
(16)	Header(19) * $(253-3)/(\rho_{max}-\rho_{min})$
(17)	$(3\rho_{max}-253\rho_{min})/(\rho_{max}-\rho_{min})$
(18)	Cell Constant Scaling Factor
(19)	100

Any density value is really equal to the value of the byte times

$$Header(16)/Header(19) + Header(17).$$

The biggest problem with DSN6 files is confusion about byte swapping differences between big and little endian computers. The bytes of the bricks present no problem but the header record does.

In the old style of DSN6 file (Frodo days) one would write the bytes in the bricks as though they were integer*2. This would cause them to be swapped in the same way as the header record and allowed the general Unix command **dd**, to convert the file from one byte swap to another.

In the new style (O days) the header record is written as though the computer is big endian, reguardless of its true endianisity.

Appendix B

Generating REFI Geometry Files from TNT Files

Any conversion of data from one format to another presents problems. The severity of the problems depends upon the amount of difference in the basic structure of the data itself. The reformatting a structure factor file would seem to be simple because everyone agrees to use integral indices and agrees that there are three of them. However even this case is not so simple. Different formats require different asymmetric units, and different sorting. There are other differences like the difference between structure factor intensities and amplitudes.

The conversion of TNT's ideal geometry data files to those of Frodo and O presents a particularly difficult problem. These file formats embody quite different views of the data required to define the conformation of a molecule.

Because of the fundamentally different representation of the data it is impossible to generate a REFI format file in TNT without some intervention on your part. It is almost certain that you will have to add additional GEOMETRY statements to your geometry definition to satisfy this program. In addition you may discover that you need to alter some existing GEOMETRY statements.

The up side of this is that you might very well uncover errors of ommission in you TNT geometry definition. TNT does not provide any assurance that the ideal geometry definition you supply is complete. In fact, it is a useful feature of TNT that you can leave out restraints for particular geometry items if you are really unsure what their value should be. The REFI format does not allow any of its values to be left undefined. If you neglected to define a particular bond angle no program in TNT will inform you of the problem. The conversion to REFI format will.

The Basic Problem

In TNT stereochemical restraints are simply listed. The order is unimportant and the completeness is unimportant. You can define bond lengths, bond angles, torsion angles, (pseudorotation angles,) planes, and chiral centers.

The REFI format contains two types of information. One is a set of pointers which define, in a rigorous fashion, the connectivity of the molecule. The other is a set of bond lengths, angles, and torsion angles.

The first difference you will notice is that TNT does not explicitly define the connectivity of the molecule. The format conversion routine builds such a table from the bond length, bond angle, torsion angle, and chiral center definitions. However, even TNT requires that all bond lengths be defined (otherwise the bonded atoms will be pushed apart because of a "bad" contact). You should never experience a problem because of an undefined bond length.

While REFI does not require that every bond angle be defined it is quite insistent that the bond angles it wants are defined. You will not be able to predict which angles are required. Convert will produce an warning message for each missing angle. You can add these angles to you TNT geometry definition file and run Convert again.

REFI has two different functions and these two functions make differing uses of the torsion angles. The usual mode of REFI is to regularize a model. When regularizing REFI only uses the torsion angle values for particular angles which have been marked as "fixed". If you have forgotten to define an torsion angle, or the angle has been defined with multiple correct values Convert will mark the angle as "variable". You will still be able to regularize your molecule but these torsion angles will not be idealized. This is only a problem for planes.

REFI uses torsion angle restraints to enforce planarity. If you have not defined the torsion angles of the planar group to be single valued, as either *cis* or *trans*, REFI will not produce a group which is flat. You will receive warning messages about each ambiguous torsion angle. You are not required to do anything about these warnings but you will get better results if you have fewer warnings.

The REFI format uses a "tree" data structure to represent the connectivity of the molecule and every tree must have a root. TNT is fairly arbitrary about its choice of root. If its choice is not satisfactory you can change it by reordering the GEOMETRY statements. Place the statements which define the bond for your root first and TNT will chose it as the root. I know of no reason to worry about the particular atom at the root so you should never have a problem with this.

Planes and Chiral Centers

Another difference between the TNT and REFI standard geometry formats is that the REFI format does not allow the definition of planes and chiral centers. This ommission does not mean that you will not have planar groups and proper chiral centers if you use REFI. These quantities can be enforced by a judicious use of torsion angles.

Imagine a benzene ring. In TNT you would state that the six carbon atoms are in a common plane. In REFI format you would specify that all the torsion angles are equal to zero. Both methods get the job done. Converting from one to the other is a problem.

If you have four atoms, which form a torsion angle, and all lie in a plane the torsion angle might be zero degrees or it might be 180 degrees. There is no way to tell if all you know about is the plane. The TNT format converter will hazard a guess but will warn you that there is a problem. The messages will be discussed later.

You can also define chirality with torsion angles. TNT will perform this conversion without bothering you.

Your biggest problem will be undefined torsion angles. The general recommendation of the authors of TNT is to leave torsion angles unrestrained during refinement. Because the angles would not be used often people simply don't define them. REFI will not allow you to get away with such laziness. You don't have to define the torsion angles for every set of four atoms – just a subset which is almost impossible to predict. The simplest way to define these angles is to attempt the conversion and read the messages. The converted will list all the undefined, but required, angles.

Most of This Belongs Somewhere Else...

but for now I am including it here.

When defining a standard geometry file you must ensure that the restraints you impose on the model are independent of each other. Usually this is simple. Bond lengths and bond angles are independent of each other in all cases. Since these types of restraints are what people are most concerned with they get lax.

The issue arrises when one introduces torsion angles and planes. For example, consider the configuration of atoms about atom CG of aspartic acid. There are three bond angles – CB-CG-OD1, CB-CG-OD1, and OD1-CG-OD2. In principle these angles could assume widely varying values and can be considered uncorrelated. When you also impose the planarity of this group you have a problem. The sum of the three angles must equal 360 degrees. This relationship allows you to calculate one angle if you know the other two. When restraining these atoms to a plane you must leave one of the angle restraints out. It doesn't really matter which one.

Similar dependences can arise with the inclusion of torsion angles in the library.

The standard TNT library is not very good at handling this problem. In the case of aspartic acid all three bond angles and the plane are defined as restraints. I suppose I should fix this in the next generation library.

Why do I bring this up in a discussion of converting a library to REFI format? Because REFI handles the problem of overdetermination of restraints in a very exact manner. You simply cannot write a REFI definition which contains too many bond angle definitions because the redundant angles have no place in the file. The format implements this restriction by specifying exactly which angles are proper and which are redundant.

The problem is that the author of the library you are using may not have had the same opinion about which angle is the redundant one. The choice is completely arbitrary. In the aspartic acid example, your library may not have a restraint defined for the angle CB-CG-OD2. The REFI format requires this angle and will ignore your definition of the OD1-CG-OD2 angle. The result is a warning that you have failed to define the CB-CG-OD2 angle. This message does not mean that the TNT library was wrong only that differing and incompatible conventions are being used.

The Messages

```
WARNING:  Required bond angle not found between atoms CD  , N   , and
          C    in group PRO
```

This warning will be issued when a bond angle definition is missing. You should add to your geometry definition the line

```
               GEOMETRY PRO ANGLE ???  ??  C, N, CD
```

substituting the proper values for the angle and its standard deviation.

```
WARNING:  Required torsion angle not found between atoms CG  , CD  , N
          , and C    in group PRO
```

This warning will be issued when a additional torsion angle definition is required. You should add

```
GEOMETRY PRO TORSION ???  ??  C, N, CD, CG
```

substituting the proper values for the torsion angle and its standard deviation.

```
WARNING:  In group HIS  the torsion angle for the atoms CB  , CG  ,
          ND1 , and CE1  is either cis or trans because all the atoms
          are in the same plane.  The REFI dictionary will assume they
          are trans.
```

This warning is issued when a torsion angle must be either *cis* or *trans* because all the atoms are in a plane. There is no way for the program to distinguish these two possibilities but I have found that *trans* is more probable. If the angle is *trans* you need do nothing. If the angle is *cis* you must enter a GEOMETRY statement similar to

```
GEOMETRY PRO TORSION 1000 5 CB, CG, ND1, CE1
```

In this particular example the angle is properly *trans* and no changes need be made.

```
WARNING:  In group HIS  the torsion angle for the atoms CG  , CD2 ,
          NE2 , and CE1X is either cis or trans because all the atoms
          are in the same plane.  The REFI dictionary will assume they
          are cis because the last atom is a ring closure atom.
```

This warning is also produced when there is an ambiguity between *cis* and *trans* but in this case the program guesses that *cis* is correct. The difference is that the last atom in the list is marked (with the appended 'X') as a "ring closure" atom. If the angle really is *trans* you must enter the GEOMETRY statement

```
GEOMETRY PRO TORSION 1180 5 CG, CD2, NE2, CE1
```

(Note that the 'X' is left off the name of 'CE1'.) If the angle is properly *cis* no changes need to be made. In this example the program's guess is correct and no action is required.

```
WARNING:  Required torsion angle not found between atoms C4A , CHB ,
          C1B , and NBX  in group PRF .  This angle could be defined
          by either the definition of a plane covering (at least) C2B
          , CHB , C1B , and NBX  or a chiral center located at C1B ,
          or failing that provide the value of the torsion angle.
```

This warning is issued when a torsion angle has not been defined but the angle could have been inferred from either a plane or a chiral center definition. You have to examine the structure of the group and chose the appropriate response. In this case all of these atoms should be in a common plane – the plane of the bacteriochlorophyll-a ring.

If neither a plane or a chiral center definition is appropriate for this group of atoms you can define a torsion angle. Usually, however, it is proper in these cases to leave the angle undefined and allow this angle to be classified as “variable”.

```
WARNING:  Required torsion angle not found between atoms NA  , MG  ,
          NB  , and C1BX in group PRF .  This angle is not the
          'primary' angle at this location.  To define this angle you
          need to define a plane or chiral center, or as a last resort
          provide the value of the torsion angle.
```

This angle is similar to the last except for its relationship to another torsion angle – here NA-MG-NB-C4B. If you want to define this angle with a chiral center or plane you must also define a single valued torsion angle for this other group. In almost all cases this is the proper response. If there is no chiral center or plane here then either leave this angle as “variable” or defined it with a single valued torsion angle.

Appendix C

Reciprocal Space Asymmetric Units

The following table lists the reciprocal space asymmetric units for all the space groups handled by TNT. The unique portion of the data for a space group is determined by the lattice class, the point group, and sometimes, the orientation.

There are two levels of description. First a box is defined. If you give the reflections in this box you will have all the data you need, but some reflections might be redundant. The conditions listed to the right of the box definition specify a more restricted collection of data, where every reflection is unique.

When a TNT program calculates structure factors it will produce a data set filling the full box. Some of the reflections in this box will be redundant. Any TNT program calculating a map from Fourier coefficients will ignore these redundant data points. When a data set is passed through the **correct** script command only the unique reflections remain. Since any refinement calculation only uses reflections that occur in both FO and FC the extra reflections in FC will have no effect.

Lattice Class	Space Group	Asymmetric Unit			Additional Restrictions
Triclinic	P1	$-\infty$	$\leq h \leq$	∞	When $h = k = 0$
	P$\bar{1}$	0	$\leq k \leq$	∞	$0 \leq l \leq \infty$
		$-\infty$	$\leq l \leq$	∞	When $k = 0$
					$0 \leq h \leq \infty$
Monoclinic-A	P2 a	0	$\leq h \leq$	∞	When $k = 0$
	P2_1 a	0	$\leq k \leq$	∞	$0 \leq l \leq \infty$
	P2/m a	$-\infty$	$\leq l \leq$	∞	
Monoclinic-B	P2 b	0	$\leq h \leq$	∞	When $h = 0$
	P2_1 b	0	$\leq k \leq$	∞	$0 \leq l \leq \infty$
	P2/m b	$-\infty$	$\leq l \leq$	∞	
Monoclinic-C	P2 c	$-\infty$	$\leq h \leq$	∞	When $k = 0$
	P2_1 c	0	$\leq k \leq$	∞	$0 \leq h \leq \infty$
	P2/m c	0	$\leq l \leq$	∞	
Trigonal-C	P3	0	$\leq h \leq$	∞	When $h = k = 0$
	P3_1	0	$\leq k \leq$	∞	$0 \leq l \leq \infty$
	P3_2	$-\infty$	$\leq l \leq$	∞	When $h = 0$
	P3/m				$k = 0$
Tetragonal-C	P4	0	$\leq h \leq$	∞	When $h = 0$
	P4_1	0	$\leq k \leq$	∞	$k = 0$
	P4_2	0	$\leq l \leq$	∞	
	P4_3				
	P4/m				

Lattice Class	Space Group	Asymmetric Unit			Additional Restrictions
Hexagonal-C	P6	0	$\leq h \leq$	∞	When $h = 0$
	$P6_1$	0	$\leq k \leq$	∞	$k = 0$
	$P6_2$	0	$\leq l \leq$	∞	
	$P6_3$				
	$P6_4$				
	$P6_5$				
	P6/m				
Orthorhombic-D	P222	0	$\leq h \leq$	∞	
	$P222_1$	0	$\leq k \leq$	∞	
	$P2_12_12$	0	$\leq l \leq$	∞	
	$P2_12_12_1$				
	Pmmm				
Trigonal-D	P312	0	$\leq h \leq$	∞	When $k = 0$
	$P3_112$	0	$\leq k \leq$	h	$0 \leq l \leq \infty$
	$P3_212$	$-\infty$	$\leq l \leq$	∞	
	P321		same		When $h = k$
	$P3_121$				$0 \leq l \leq \infty$
	$P3_221$				
	P3mm				
Tetragonal-D	P422	0	$\leq h \leq$	∞	
	$P4_212$	0	$\leq k \leq$	h	
	$P4_122$	0	$\leq l \leq$	∞	
	$P4_12_12$				
	$P4_222$				
	$P4_22_12$				
	$P4_322$				
	$P4_32_12$				
	P4mm				

Lattice Class	Space Group	Asymmetric Unit	Additional Restrictions
Hexagonal-D	P622	$0 \leq h \leq \infty$	
	$P6_122$	$0 \leq k \leq h$	
	$P6_222$	$0 \leq l \leq \infty$	
	$P6_322$		
	$P6_422$		
	$P6_522$		
	P6mm		
Cubic-T	P23	$0 \leq h \leq \infty$	When $h <> k$
	$P2_13$	$0 \leq k \leq h$	$0 \leq l \leq h-1$
	P2/m 3	$0 \leq l \leq h$	
Cubic-O	P432	$0 \leq h \leq \infty$	
	$P4_132$	$0 \leq k \leq h$	
	$P4_232$	$0 \leq l \leq k$	
	$P4_332$		
	P4/m 32		

Appendix D

Real Space Asymmetric Units

This table lists the real space asymmetric units for all the space groups handled by TNT. The unique portion of the data for a space group is determined by the lattice class, the point group, and sometimes, the orientation.

Because the FFT algorithm cannot handle all types of symmetry the Fourier synthesis calculations for some space groups are actually performed in a lower symmetry group. This requires that the real space asymmetric unit be larger by a factor of the multiplicity of the symmetry being ignored. For example, direct three-fold axes can never be used to speed the calculation of an FFT. Therefore space group P23 is handled as though it were space group P222. The amount of map required is a factor of three larger that that expected for this group.

The "Additional Restrictions" column lists the additional factors that the sampling rate must contain. No sampling rate, in any direction, may have a prime factor larger than 17. The smaller the largest prime factor is the more efficient the FFT calculations will be.

Lattice Class	Space Group	Asymmetric Unit			Additional Restrictions
Triclinic	P1	0	$\leq x <$	nx	nx divisible by 2
	P$\bar{1}$	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z <$	nz	
Monoclinic-A	P2 a	0	$\leq x <$	nx	nx divisible by 2
	P2_1 a	0	$\leq y <$	ny	ny divisible by 2
	P2/m a	0	$\leq z \leq$	nz/2	nz divisible by 2
Monoclinic-B	P2 b	0	$\leq x <$	nx	nx divisible by 2
	P2_1 b	0	$\leq y <$	ny	ny divisible by 2
	P2/m b	0	$\leq z \leq$	nz/2	nz divisible by 2
Monoclinic-C	P2 c	0	$\leq x \leq$	nx/2	nx divisible by 2
	P2/m c	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z <$	nz	
	P2_1 c	0	$\leq x <$	nx	nx divisible by 2
		0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/2	nz divisible by 2
Trigonal-C	P3	0	$\leq x <$	nx	nx divisible by 2
	P3/m	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z <$	nz	
	P3_1	0	$\leq x <$	nx	nx divisible by 2
	P3_2	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z <$	nz/3	nz divisible by 3

Lattice Class	Space Group	Asymmetric Unit	Additional Restrictions
Tetragonal-C	P4	$0 \leq x \leq$ nx/2	nx divisible by 2
	P4/m	$0 \leq y <$ ny	ny divisible by 2
		$0 \leq z <$ nz	
	$P4_1$	$0 \leq x <$ nx	nx divisible by 2
	$P4_3$	$0 \leq y <$ ny	ny divisible by 2
		$0 \leq z <$ nz/4	nz divisible by 4
	$P4_2$	$0 \leq x \leq$ nx/2	nx divisible by 2
		$0 \leq y <$ ny	ny divisible by 2
		$0 \leq z <$ nz/2	nz divisible by 2
Hexagonal-C	P6	$0 \leq x \leq$ nx/2	nx divisible by 2
	$P6_3$	$0 \leq y <$ ny	ny divisible by 2
	P6/m	$0 \leq z <$ nz	
	$P6_1$	$0 \leq x <$ nx	nx divisible by 2
	$P6_5$	$0 \leq y <$ ny	ny divisible by 2
		$0 \leq z <$ nz/6	nz divisible by 6
	$P6_2$	$0 \leq x \leq$ nx/2	nx divisible by 2
	$P6_4$	$0 \leq y <$ ny	ny divisible by 2
		$0 \leq z <$ nz/3	nz divisible by 3
Orthorhombic-D	P222	$0 \leq x \leq$ nx/2	nx divisible by 2
	$P2_12_12$	$0 \leq y <$ ny	ny divisible by 2
	Pmmm	$0 \leq z \leq$ nz/2	nz divisible by 2
	$P222_1$	$0 \leq x <$ nx	nx divisible by 2
	$P2_12_12_1$	$0 \leq y <$ ny	ny divisible by 2
		$0 \leq z \leq$ nz/4	nz divisible by 4

Lattice Class	Space Group	Asymmetric Unit			Additional Restrictions
Trigonal-D	P312	0	$\leq x <$	nx	nx divisible by 2
	P321	0	$\leq y <$	ny	ny divisible by 2
	P3mmm	0	$\leq z \leq$	nz/2	nz divisible by 2
	$P3_112$	0	$\leq x <$	nx	nx divisible by 2
	$P3_12_1$	0	$\leq y <$	ny	ny divisible by 2
	$P3_212$	0	$\leq z \leq$	nz/6	nz divisible by 6
	$P3_221$				
Tetragonal-D	P422	0	$\leq x \leq$	nx/2	nx divisible by 2
	P4mm	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/2	nz divisible by 2
	$P4_122$	0	$\leq x <$	nx	nx divisible by 2
	$P4_12_12$	0	$\leq y <$	ny	ny divisible by 2
	$P4_322$	0	$\leq z \leq$	nz/8	nz divisible by 8
	$P4_32_12$				
	P4222	0	$\leq x \leq$	nx/2	nx divisible by 2
	$P4_22_12$	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/4	nz divisible by 4
Hexagonal-D	P622	0	$\leq x \leq$	nx/2	nx divisible by 2
	$P6_322$	0	$\leq y <$	ny	ny divisible by 2
	P6mm	0	$\leq z \leq$	nz/2	nz divisible by 2
	$P6_122$	0	$\leq x <$	nx	nx divisible by 2
	$P6_522$	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/12	nz divisible by 12
	$P6_222$	0	$\leq x \leq$	nx/2	nx divisible by 2
	$P6_422$	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/6	nz divisible by 6

Lattice Class	Space Group	Asymmetric Unit			Additional Restrictions
Cubic-T	P23	0	$\leq x \leq$	nx/2	nx divisible by 2
	P2/m 3	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/2	nz divisible by 2
	$P2_13$	0	$\leq x <$	nx	nx divisible by 2
		0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/4	nz divisible by 4
Cubic-O	P432	0	$\leq x \leq$	nx/2	nx divisible by 2
	P4/m 32	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/2	nz divisible by 2
	$P4_132$	0	$\leq x <$	nx	nx divisible by 2
	$P4_332$	0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z \leq$	nz/8	nz divisible by 8
	$P4_232$	0	$\leq x \leq$	nx/2	nx divisible by 2
		0	$\leq y <$	ny	ny divisible by 2
		0	$\leq z <$	nz/4	nz divisible by 4

Index